12^{00}

ARRL/CRRL AMATEUR RADIO

9TH COMPUTER NETWORKING CONFERENCE

LONDON, ONTARIO CANADA

SEPTEMBER 22, 1990

ARRL/CRRL AMATEUR RADIO

9TH COMPUTER NETWORKING CONFERENCE

COORDINATORS:
Harry MacLean, VE3GRO
David Toth, VE3GYQ
Paul L. Rinaldo, W4RI

HOSTS:
Canadian Radio Relay League (CRRL)
American Radio Relay League (ARRL)

Canadian Radio Relay League
London, Ontario Canada N5Y 4J9

American Radio Relay League
Newington, CT USA 06111

Copyright 1990 by

The American Radio Relay League, Inc.

International Copyright secured

This work is Publication No. 132 of the Radio Amateur's Library, published by the League. All rights reserved. No part of this work may be reproduced in any form except by written permission of the publisher. All rights of translation are reserved.

Printed in USA

Quedan reservados todos los derechos

$12.00 in USA

ISBN: 0-87259-337-1

First Edition

Foreword

The American Radio Relay League takes pride in publishing papers for this Ninth joint CRRL/ARRL Amateur Radio Computer Networking Conference.

Several of these papers reflect the growing interest in-and use of- packet radio satellite. Six amateur packet satellites were launched in the spring of 1990, with several more under construction or on the drawing boards. Also prominent in this year's papers are means of attacking the channel-access problems, as well as continuing development of networking protocols and implementations.

As in previous conferences, all papers in these proceedings are unedited and solely the work of the authors.

David Sumner, K1ZZ
Executive Vice President

Newington, Connecticut
September, 1990

Trademarks appearing in these papers are:

AEA is a trademark of Advanced Electronic Applications, Inc.
ANSI is a registered trademark of the American National Standards Institute
Apple is a trademark of Apple Computer Inc.
AppleShare is a trademark of Apple Computer Inc.
AppleTalk is a trademark of Apple Computer Inc.
CompuServe is a trademark of CompuServe Inc.
Data Engine is a trademark of Kantronics Co., Inc.
DVR 2-2 is a trademark of Kantronics Co., Inc.
IBM is a registered trademark of International Business Machines
LocalTalk is a trademark of Apple Computer Inc.
Macintosh is a trademark of Apple Computer Inc.
MS-DOS is a registered trademark of Microsoft Corporation
NET/ROM is a trademark of Software 2000 Inc.
Tandy is a trademark of Tandy Corp.
UNIX is a registered trademark of AT&T
Z80 is a trademark of Zilog Corp.

TABLE OF CONTENTS

9600 Baud Operation

Interfacing the Kantronics DataEngine and DVR2-2 Transceiver with the G3RUH External Modem

by Phil Anderson, WØXI, and Karl Medcalf, WK5M

May 21, 1990

Lawrence, Kansas

Over the last month or so, we've conducted a number of on-the-air tests with the G3RUH modem[1] in combination with the Kantronics DataEngine and the Kantronics DVR2-2 2-meter data/voice transceiver at 9600 baud. After the usual initial false starts we found the combination to work quite well. We passed a large number of 10K files in transparent mode from WK5M's station to my PBBS (mailbox in my DataEngine) without error. Additionally, we performed some informal bit error rate (BER) tests on the link, finding results comparable to those listed in Steve Goode's (K9NG) paper[2].

While additional detailed BER testing is called for to obtain a firm measure of the capability for 9600 operation by the combination listed above, additional on-the-air testing by more stations is desirable. Hence, we present the details of interfacing the DataEngine (DE) and DVR2-2 with the external G3RUH modem (RUH) to facilitate these tests.

Building the 9600 baud station consists of wiring four interfaces:

- your computer to the DataEngine,
- installing internal disconnect header jumpers within the DataEngine,
- DataEngine to RUH, including a push-to-talk circuit,
- and RUH to DVR cabling.

The combination is shown in Figure 1. If you have a DataEngine already, the computer to DataEngine interface/cabling is probably operational. Since the RUH modem assumes push-to-talk (PTT) circuitry is provided by the TNC, you'll have to add that somewhere. On our initial tests, we wired a PTT circuit directly to the RUH printed circuit (pc) board; however, at this point, we'd suggest that you consider adding the PTT to the internal wiring of the DataEngine. The PTT circuit is depicted in Figure 2.

FIGURE 1. 9600 baud station: Kantronics DataEngine, G3RUH external modem and the Kantronics DVR2-2 transceiver

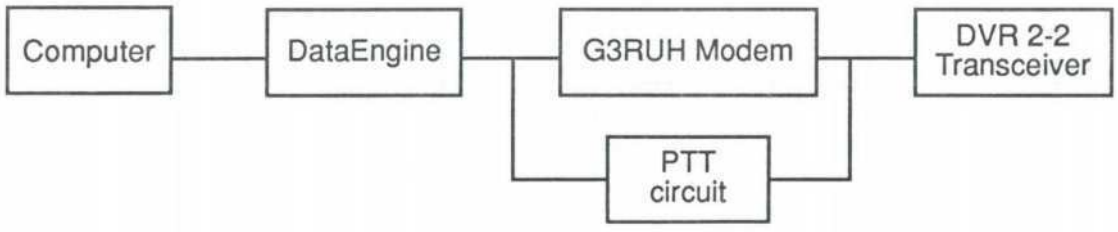
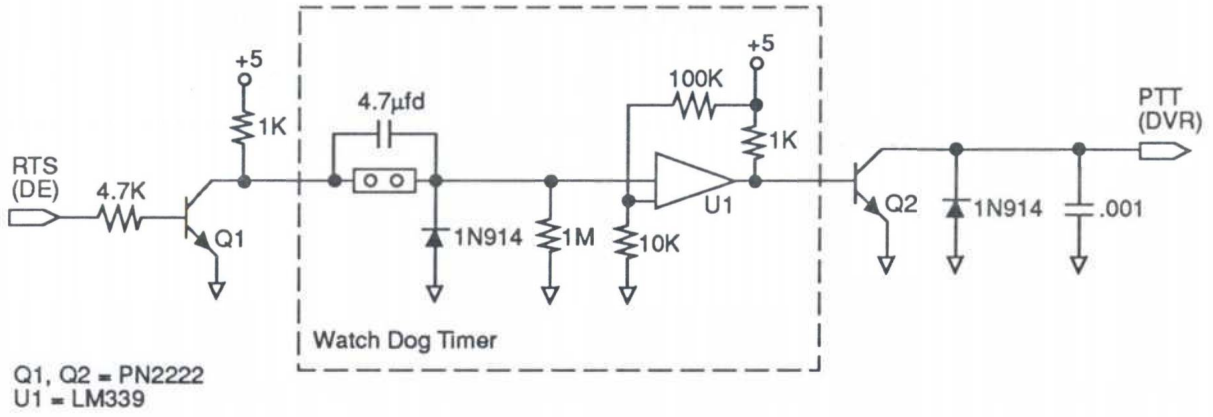


FIGURE 2. Addendum Push-To-Talk and Watchdog circuitry[8]



DataEngine Interfacing

The DataEngine was designed to accept a wide range of modems. Two pairs of disconnect headers on the main pc board can accommodate two modems within the case environment, assuming physical compatibility. (A family of Kantronics' designed modems are planned for internal installation – one DE1200 is now shipped with each DE). However, existing modems, such as the RUH, are already on a pc board and hence must be interfaced outside the DataEngine case. In these cases, jumpers must be installed inside the DE at the disconnect header pairs to match external modem characteristics/needs. For more information, refer to the DE Operator's Manual[3]. Then, a cable must be prepared to go between the DE DB-15 back panel connector and the RUH headers.

Internal disconnect headers

Figure 3 denotes the necessary jumpers required between the two internal disconnect headers. You may wire either port 1 or port 2, or if you like you may wire both ports for 9600 baud operation! The DataEngine is fully capable of handling simultaneous operation of both ports at once.

FIGURE 3. Internal jumpers required for a 'Type C Modem' external attachment to the DataEngine.

INT pin	EXT pin	Comment
3	3	TXD, transmit data (to external)
4	4	RXD, receive data (from external)
6	6	RTXC, receive clock (from external)
7	7	RTS, push-to-talk * (to external)
9	9	DCD, carrier detect (from external)
13	13	TOUT2, 16X T clock (to external)
15	19 (gnd)	MS0, modem addr jumper (internal)
16	19 (gnd)	MS1, modem addr jumper

* If the PTT circuit is wired internal to the DE, then drive the PTT circuit from pin 7 of the INT header and pass the PTT output to pin 7 of the EXT header. If you wire the PTT on the RUH modem board, then just jump INT 7 to EXT 7. See Figure 2 for suggestions.

Each pair of disconnect headers is labeled INT (internal) and EXT (external). For example, on the pc board you'll note the port 1 headers are labeled A INT1 and A EXT1. Orienting your DE pc board with the copyright notice in the upper left-hand corner and the power connector to the lower right, the port 1 headers are located immediately to the left of the power connector at the back panel. Both the internal and external headers contain 20 pins, and these are numbered odd on the left and even on the right. Pin 1 is at the top left in each case.

For our experiments, we used pin-crimped wire jumpers such as those found in RS-232 break-out boxes. However, an internal jumper pc board is planned if enough interest is generated, including the required additional PTT circuitry.

Signals

The DE must supply TXD, TOUT2 (a 16 times transmit clock), RTS (push-to-talk (PTT)), and +12VDC to the RUH modem. The DE must receive RXD, RTXC (receive clock), and DCD from the modem. Additionally, the DE must be told the kind of modem that is being attached, in this case a 'Type C Modem'. This is done by grounding pins 15 and 16 to pin 19, ground.

A Type C modem, for the DE, is defined as one that requires a 16X transmit clock from the TNC but supplies to the TNC a receive clock at the data rate.

DataEngine to RUH interface

The RUH pc board has three disconnect headers on it, P1, P2, and P3. Interfacing between the modem and the DE involves P1 and P2. Make a cable going from the DE DB-15 and "Y'd" off to P1 and P2 at the modem as listed in Figure 4. The modem kit includes pins and a set of plastic headers to hold them.

Header P2 includes the following signals: TXD, RXD, DCD, TXC, RXC, and GND. P1 includes +12VDC and GND. The +12VDC from the DE has enough current capacity to handle the modem requirements, so we've listed pin 13 of the DB-15 connector that carries the +12VDC. Jumper JP1 or JP2 within the DE must be set on the center pin and A so that pin 13 does indeed deliver the +12VDC. CAUTION: Use a well regulated supply for the DE and RUH combination. We found a 'raw' wall adapter with poor regulation caused our system to fail.

FIGURE 4. DataEngine to G3RUH modem interface

DE DB-15	RUH P2 header	Comment
1	1	TXD, transmit data
2	4	RXD, receive data
4	6	RTXC, receive clock
5		PTT, see text
7	5	DCD, data carrier detect
11	2	ground
12	3	TOUT2, 16X transmit clock
13	P1-pin2	+12VDC (set jumper JMP2.)
11	P1-pin1	ground

For more detail, refer to the DE schematic. Pin-outs at the DB-15 DE connector will differ by modem. Pin-outs for this experiment WILL NOT correspond with those for the DE1200 modem. The modems, by nature, are different.

RUH to Kantronics DVR2-2 Transceiver Interface

The cable must run from header P3 of the RUH to the data port of the DVR. The DVR requires PTT, TXD and ground, and delivers RXD (unsquelched) and DCD (not used in this application). Refer to the DVR Operator's Manual[4] and Figure 5 for more detail.

We recommend attaching a speaker to the audio-out jack of the DVR. You'll get a kick out of listening to the 9600 'hiss' if you've not heard it before.

The Push-To-Talk Addition

The PTT signal is assigned pin 5 on the DE DB-15 port connector. If the PTT circuitry is added internally to the DE, its source from the processor is at pin 7 on the INT header. Pin 5 of the back panel connector (DB-15) is pc-wired directly to pin 7 of the EXT header. So, have the output of the PTT circuit wired directly to EXT header, pin 7.

If you plan to add the PTT circuit on the extra room and holes provides on the RUH board, as we did, then just jumper INT pin 7 to EXT pin 7 which in turn is directly pc-wired to pin 5 of the back panel DB-15 as above. Either way, the resulting PTT output must be connected to pin 3 of the DVR data port DB-9 connector. Again, see Figure 2 for choices.

DVR Transceiver Modifications

As it turns out, the drive level of the RUH is a bit high for the data input of the DVR. The DVR TXD line expects to see a drive level of about 50 MV P-P. Both VE3ZAV[5] in Canada, and G4HCL[6] in England reported this. Both reduced TX audio drive for successful 9600 baud operation.

You can reduce the drive at the modem, but we found it easier and neater to put the resistor divider inside the DVR – change resistor R43 to 47K.

Additionally, the RUH desired a bit more discriminator drive than the production DVR delivers. So, also bypass resistor R30 with a jumper. 150 MV p-p instead of 50 MV p-p is then available for the modem, which incidentally calls for a minimum of 50 mv drive.

Tuning up

After preparing all cabling and checking each cable, you'll want to check that your DVR2-2 radios are on frequency and that the proper amount of deviation is generated with each modem-radio pair. We used an IFR service monitor to set our pairs.

First, we keyed the transmitter and adjusted L22 to confirm that the DVR was on frequency. Then we set the DataEngine to a TXD of 255 and transmitted unproto

FIGURE 5. Kantronics DVR2-2 to RUH cabling

DVR DB-9	RUH P3 header	Comment
1	1	TXD, transmit audio
2	NC	DCD, (not used this app, internal)
3		PTT, see text
5	4	RXD, received audio (unsquelched)
6	2	ground
8	NC	speaker audio

packets through the modem-DVR pair and adjusted the modem drive such that the service monitor measured 3 KHz deviation. You could also set deviation using the BERT feature of the modem.

Additionally, we tweaked up the output of the discriminator of the DVRs using a standard scope. When no RF signal is present you can balance the random output so that the pattern is 'balanced'. Alternatively the service monitor can be used to optimize the SINAD reading.

Packet Parameter Considerations

With the DVR radios at both ends, start with a TXD of 5. This should be plenty long enough. Start operation by exchanging unproto packets. In this way, if either end has a problem, it can be quickly located. There is nothing magic about the operation of the DataEngine, G3RUH, DVR combination. They do work. So if you run into difficulty, check the basics: cabling, power supply, TNC parameters, right port (that one caught us once for a half-hour).

Enjoy and let us know of your experiences.

Since we originally published this paper, Kantronics has designed and is now producing an internal disconnect jumper board which provides the watchdog timer and PTT circuits described earlier in this paper. The board plugs onto the two disconnect headers inside the Data Engine, and provides all required signals from the processor to the DB-15 connector on the rear panel.

In addition, the jumper board has jumpers which allow you to select the modem TYPE (A, B, or C). Currently, this covers all known external modems including the G3RUH, HAPN, K9NG, and PSK-1 to mention a few. The watchdog timer is set for 15 seconds, but this may be changed be replacing the 4.7 micro-farad capacitor with a different value if required. The watchdog timer may be disabled by placing a jumper in the appropriate position.

References

1. Miller, James R. G3RUH. "9600 baud Packet Radio Modem Design", Proceedings of 7th ARRL Computer Networking Conference, Oct. 1988, pps. 135-140.

2. Goode, Steve. K9NG. "Modifying the Hamtronics FM-5 for 9600 BPS packet operation". In proceedings of the Fourth ARRL Amateur Radio Computer Networking Conference, March 30, 1985, San Francisco.

3. Operator's Manual, Kantronics Data Engine. Appendix: Connecting Other Modems to the Data Engine.

4. Operator's Manual, Kantronics DVR2-2. Data port DB-9 connector.

5. Varga, Anthony, VE3ZAV, 73 Beechbank Cresc., London, Ontario, Canada.

6. Lorek, Chris. G4HCL. "DVR2-2 2M PACKET TX". Ham Radio Today magazine, England, March 1990.

7. Kantronics Co., Inc., 1202 E 23rd St., Lawrence, KS 66046 913-842-7745, BBS 913-842-4678, FAX 913-842-2021.

8. Installation Manual, Kantronics, KAM.

DIGITAL REGENERATOR MODIFICATION FOR THE TNC-2A

William A. Beech, NJ7P

1953 Santa Catalina
Sierra Vista, AZ 85635
INTERNET: bbeech@huachuca-emh8.army.mil

Jack Taylor, N7OO

RR-2 Box 1640
Sierra Vista, AZ 85635

ABSTRACT

Others have pointed out the network efficiency advantage of using duplex regenerators as compared to simplex digipeaters (Ref 1 & 4). However, the use of packet duplex regenerators entails added expense which may be the reason they have not found common usage in LAN and backbone systems. In addition to the expense of a duplexer for in-band operation, previous implementations of regenerators have required either dual TNCs or the use of a high quality land line modem as well as a TNC. This paper describes a way to make multiple use of a TNC in this application, with a realized savings in cost and simplification in equipment configuration. Furthermore, if the regenerator was set up for cross-band operation, the expense then would be comparable to a simplex digipeater since a duplexer would not be required.

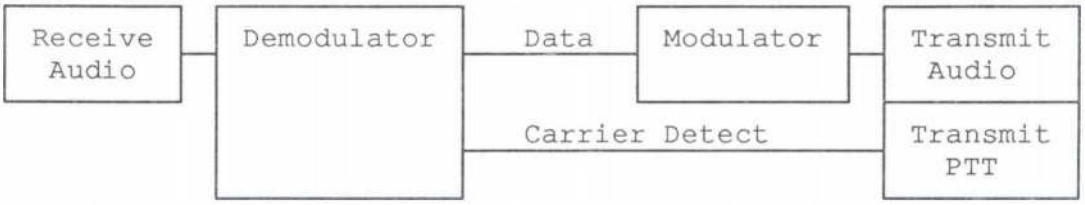
This paper describes the conversion of a TNC-2A to serve as both a digital regenerator and a network node. The conversion has been in use on the 144.51-145.11 SVALAN:N7OO-4 repeater/node for several months.

INTRODUCTION

The necessary parts required to perform the functions of both a node and a digital regenerator are available on the standard TAPR TNC-2A. This includes the TNC-2A clones which incorporate the modem disconnect header. The only additional parts are a 20-pin header and plug and the 4053 IC. The following discussion will be referenced to the MFJ 1270B TNC-2A clone. The modem disconnect header provides the interface between these components and the simple circuit required for the conversion. The TNC had the NETROM node changes, the DCD modification, and the modem disconnect header added prior to this conversion (Ref 2 & 3).

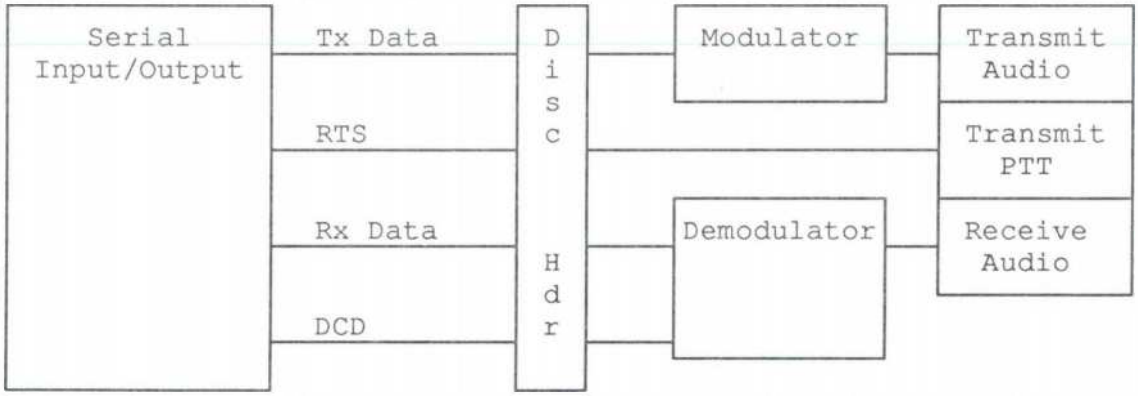
THEORY OF OPERATION

A digital regenerator requires the output of the demodulator to feed the modulator and the data carrier detect (DCD) signal to trigger the push-to-talk (PTT) on the transmitter (See Figure 1).



Digital Regenerator
Figure 1.

In a normal TNC the serial input/output (SIO) transmit data (TxD) line feeds the modulator while the request to send (RTS) line triggers the PTT of the transmitter. The demodulator drives the SIO receive data (RxD) line while the DCD is sensed by the SIO carrier detect (CD) line (See Figure 2). These signals are available at the modem disconnect header.



TNC-2A Data Flow Diagram
Figure 2.

To provide both functions, the modulator input and PTT control must be switched between the SIO signals and the demodulator signals. Since the TNC software will not allow the SIO to transmit while there is CD present, using the CD signal from the demodulator to switch the lines to the modulator is acceptable.

With this circuit, the modulator will transmit all packets seen by the demodulator. The packets are also sensed by the SIO to see if they are addressed to the node. The node can transmit through the modulator whenever the channel is clear.

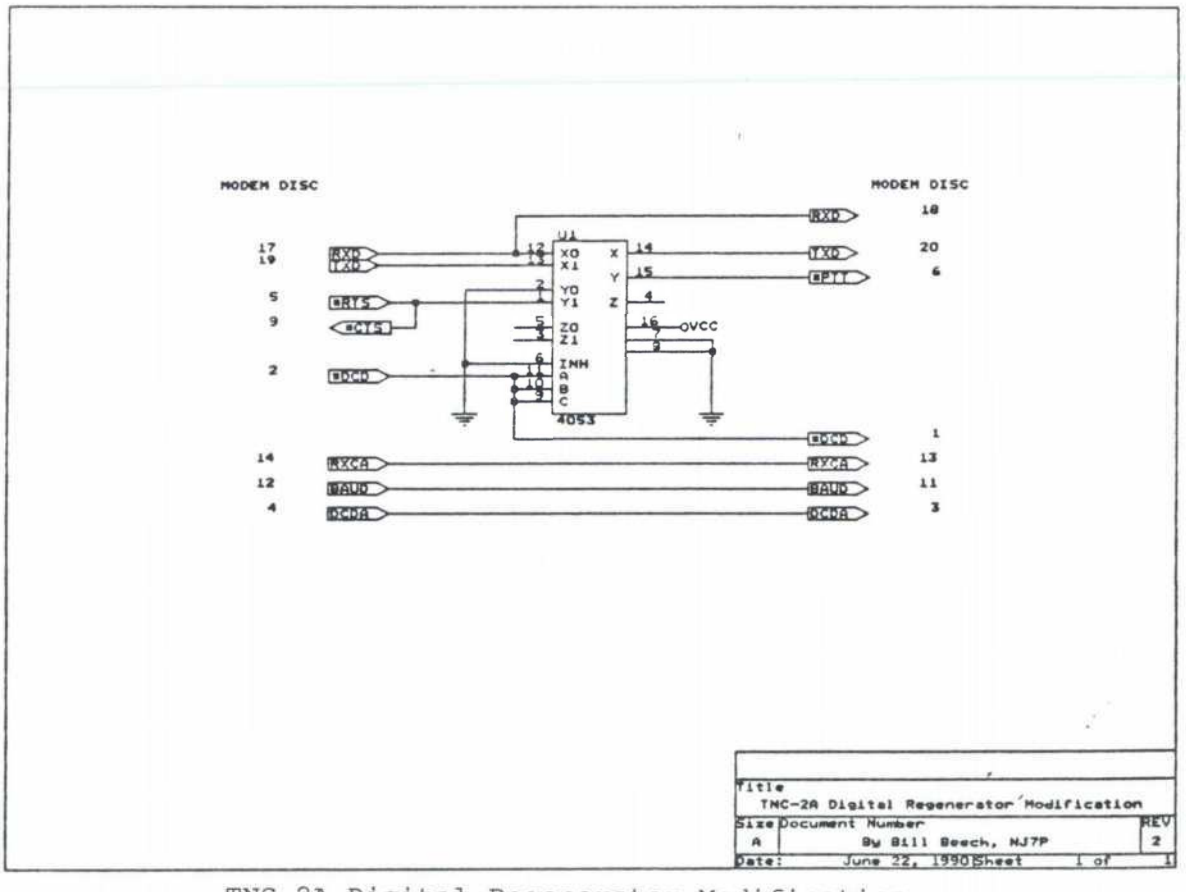
Originally a 74LS157 quad 2-input multiplexer was used in this conversion. It was found to provide an unacceptable load to the demodulator when exposed to operating temperatures in excess of 100 degrees F. The currently used CMOS 4053 triple 2-channel multiplexer has operated satisfactorily in a high temperature environment.

The modified TNC can either be used in a multinode stack or in a stand alone mode. The unit described here is currently operating in a multiple node stack. The node has performed satisfactorily for several months, once the temperature loading problem noted above was rectified.

CONVERSION

Install the modem disconnect header by cutting the pin to pin circuit trace jumpers (e.g 1-2, 5-6, 9-10, 11-12, 13-14, 17-18 and 19-20) on the J4 modem disconnect header at the underside of the board. Install the 20-pin header plug on the pads on the top of the board.

The 4053 is mounted piggyback on U9, a 74HC14. Power (4053-16) and ground (4053-8) are connected to pins 14 and 7 of U9 respectively. The remaining pins of the 4053 are bent out at a 45 degree angle. Signal lines are brought out on three inch long wires from the modem disconnect header plug. Three pairs of the wires from the header plug must be connected together. Four wires from the header plug are not connected. The rest of the wires are connected to the 4053 as noted on the diagram in figure 3.



TNC-2A Digital Regenerator Modification
Figure 3.

OPERATIONAL TEST

It is recommended the standard TAPR TNC modem tone alignment be conducted prior to placing the unit into operation. This modification converts the TNC for duplex operation. If connected to a simplex radio, all valid incoming packets will attempt to be regenerated on the channel. Don't do that! We tried it and it made a great chatter test of the PTT. Instead, connect the TNC (standard audio/PTT configuration) to a radio capable of duplex operation. The SVALAN repeater radio is a Motorola MICOR modified such that A+ is applied continously to the oscillator and all stages except

for one driver. The TNC PTT line directly keys the driver, resulting in a TXD of approximately 5 Milliseconds. Adjacent channel splatter was not detected on a spectrum analyzer. Checkout consists of observing normal operation when connecting either through the repeater or to the node.

This modified TNC should work well with any duplex radio and is not dependent on the MICOR. The switching time is not critical if users set the AXDELAY parameter to match the keyup time. We have used AXDELAY of 0 for the MICOR with no problems.

The TNC can be restored to original operation simply by removing the modem disconnect header plug and reconnecting the disconnect header jumpers with jumper blocks. U9 can be replaced or the 4053 removed from it to restore the TNC.

SUMMARY

Network improvements have not kept pace with the growth of user developments. Duplex digital regeneration techniques have demonstrated worthwhile improvements in throughput, as compared to simplex operation. This paper describes one approach to controlling a duplex regenerator/node by adding a simple low cost modification to a standard TNC-2.

References

1. "A Duplex Packet Radio Repeater Approach to Layer One Efficiency," Robert Finch, N6CXB, ARRL Sixth Computer Networking Conference Proceedings, August 1987.

2. "Installation," Software 2000, Inc, NET/ROM Version 1.3 Documentation, September 1987.

3. "DCD Modification for the TAPR TNC-2," Eric Gustafson, N7CL, TAPR Packet Status Register, August 1988.

4. "A Duplex Packet Radio Repeater Approach to Layer One Efficiency, Part Two,"Scott Avent, N6BGW and Robert Finch, N6CXB, ARRL Seventh Computer Networking Conference Proceedings, October 1988.

Considering Next-Generation Amateur Voice Systems

Jon Bloom, KE3Z

ARRL Laboratory

ABSTRACT

Next-generation voice systems are a logical outgrowth of high-speed networking. Replacement of existing systems (repeaters), which is needed to address the current spectrum-use problems, must be preceded by design of appropriate multiple-access systems. Some of the key issues to consider are analog vs. digital modulation schemes and the types of multiple access arrangements.

1. The role of computer networking in voice communications

It may at first seem odd to give a paper about next-generation voice systems at a conference devoted to computer networking. But this reflects a trend: computer networking is becoming less about computers and more about applications. That is, the availability of high-speed digital networks evokes new classes of applications, ones that have little or nothing to do with classical "computer networking." For this conference to consider the subject of voice communications systems is simply a recognition of that reality.

Most of the communication carried across existing amateur digital systems is text. But most of the communication amateurs *do* is by means of speech. This is hardly surprising; most human communication is by way of speech. So the predominant amateur communication systems in use today are those that are designed for voice communications. On the VHF and higher frequencies, analog voice repeaters are the principal means of communication. The underlying premise of this paper is that speech will continue to be the means of choice for most amateur communication. The issue to be addressed, then, is in what way the advent of high-speed digital capabilities can and will change the way in which voice communication is accomplished.

2. Limitations of existing systems

Present VHF voice systems[1] use analog FM single-channel-per-carrier (SCPC) transmissions. This is a technology developed in the period following World War II. Except for implementation details (the use of solid-state circuitry, for example), this technology has remained essentially unchanged since the 1960's. While the fact that a technology is more than twenty years old doesn't count against it, a twenty-year-old technology that hasn't changed perceptibly in that period can be assumed to be mature: little further improvement can be obtained by refining that technology. If technological improvements are needed in amateur voice communications, it seems unlikely that analog FM SCPC systems will provide them. Certainly the experience of other services, such as Land Mobile, which have strong economic incentives for the improvement of the technology, indicate that a technological plateau has been reached.

---

[1] This paper neglects HF voice circuits. While HF has a role in providing long distance point-to-point circuits, it has little foreseeable role in a general-purpose voice system.

Are improvements in voice systems needed? Consider these attributes of existing repeater systems:

- *Poor frequency reuse of voice channels.* Although amateur frequency reuse has yet to make full use of existing techniques (CTCSS, for example), applying such methods would cause only an incremental improvement. If a large number of (codeless?) licensees are to be accommodated in existing spectrum, more spectrum efficiency must be obtained.[2] Frequency reuse is a key to efficient use of the spectrum, but existing wide-area repeaters that can tolerate no interfering signals at levels above the receiver noise floor are near-worst-case systems with regard to frequency reuse.
- *Idle channels.* In major urban areas of the US, at least one of the available amateur VHF bands, and often two or three bands, are full of repeaters. That is, in the segments of the band reserved for repeaters, every channel is occupied. But many of these repeaters sit idle most of the day, coming to life only during "drive time." While peak loading performance is an important measure of any communications system, there are better ways to deal with it than having idle channels for most of the day.
- *Encroachment of packet.* To date, packet radio has made use of narrow-bandwidth systems. In areas where voice repeaters "own" all of the existing repeater allocations, packet is relegated to use of simplex techniques. At present no efficient simplex channel-access mechanisms are in widespread use. Because of this, packet users are making use of more channels to achieve lower per-channel loading, or are using repeater pairs for duplex packet systems. This, plus the push to higher speeds and concomitant higher bandwidths, is beginning to cause friction between packet and voice users. Packet systems are evolving, however, which provides an opportunity to address part of the problem in the design of new packet systems.

But at present, voice systems are *not* evolving. To address the issues noted above, amateurs need to design and implement new voice systems.

3. Amateur vs. cellular

Amateurs are not the only ones looking to the development of new voice communication systems. Cellular phone services (and others) are evolving designs to meet their voice communication needs. Since huge amounts of research and engineering are being devoted to these efforts, it might seem that amateur system designers should wait and benefit from these efforts by applying the techniques developed for cellular systems. But there are some striking differences between the amateur environment and the commercial mobile communications environment. These differences may mean that designs optimal for commercial service are anything but optimal for amateur service. Some of the key differences:

- The commercial systems will handle primarily station-to-station communications. Little if any broadcast or party-line facilities will be provided; there isn't much need for it. But the amateurs do need such services. The ability of amateurs to call CQ is important. Any type of communication that is point-to-multipoint in nature, or in which the initiating user may not know the identity of the station he is calling, requires a class of service that commercial systems are not being designed to optimize, if indeed they offer it at all.

---

[2] The oft-mentioned "moving to higher bands" solution is illusory. Microwave bands hold great promise for point-to-point circuits, but mobile operation becomes difficult at higher frequencies, and mobile operation comprises a significant part of amateur voice use.

- One of the factors driving the design of commercial systems is the need for security. Encryption of voice signals is highly desirable in a commercial radio environment. Amateurs have an exactly opposite requirement: amateur systems must be capable of being monitored. More, they should *encourage* monitoring in order that self-policing be effective.
- The economic base for the construction of amateur systems is significantly different from that for commercial systems. While commercial operators (presumably) can afford to use a large number of stations to achieve near-total coverage of a given area, this often is not the case for amateurs. In areas of hilly terrain or urban, it is difficult to avoid regions of poor coverage, and amateurs are unlikely to be able to resort to large numbers of cell sites to get around the problem.
- While commercial systems are highly reliable, the need for amateur communications is greatest in precisely those circumstances where commercial systems fail. In such circumstances, a tightly-coupled, centrally controlled system is likely to be unworkable.

The economic factors and the reliability factors argue for wide-area coverage systems. Monitoring of amateur communications will be easier in such an environment, as will point-to-multipoint applications. This is philosophically at odds with the commercial cellular approach. How much of commercial practice will be transferable to amateur systems is an open question, but it seems likely that the amateur architectures will differ from the commercial in more than name.

4. Future Amateur Voice System Capabilities

Let us assume that the locations of stations won't change significantly: repeaters will still be located at heights that provide wide-area coverage, fixed amateur stations obviously won't move, and mobiles will be... mobile. Only the characteristics of the stations will change. How must the characteristics of these stations change in order to solve the kinds of problems described above, and what useful additional services can be included in next-generation systems?

To accommodate increased traffic, more channels need to be available.[3] From an economic standpoint, it makes sense to occupy more channels not by putting up more repeaters, but by making existing repeaters service more channels (i.e., repeaters will become multiple-access systems). At the same time, a capability for full-duplex voice communication is highly desirable. Finally, repeaters should be able to operate together in a network (using the high-speed networks under development now) to increase communications range.

Fixed and mobile stations should include both half- and full-duplex capability. They should be able to communicate via repeaters or directly, just as they can now. New applications such as store-and-forward (voice-mail), "call waiting" etc. can be designed into the system. For mobiles, especially, integration of data and voice capability into one unit would be useful.

---

[3] In this context, "channel" does not (necessarily) refer to a contiguous block of spectrum that carries a circuit signal. Rather, we use a broader definition of channel: a spectrum resource—a bandwidth for some period of time—that carries a single circuit.

5. Digital vs. analog systems

One of the key questions in designing future voice systems is whether to use analog FM techniques or digital techniques between the repeaters and the user stations. For a variety of reasons, it appears that digital is the way to go. Some of the advantages of digital techniques are:

- *Error detection and correction capability.* Digital systems can potentially detect errors in received data. Error correction can in turn be implemented using FEC or ARQ techniques. This capability could be most usefully exploited in adaptable systems, wherein degradation of the RF channel would cause the use of error handling. (Error control would increase the needed bandwidth, so it probably would not normally be used. Potentially, several steps of increasingly capable error correction could be available, trading bandwidth for channel performance as the need arises.)
- *Data transparency.* In analog voice systems, any digital data must be converted to voice-bandwidth analog signals. The resulting system, optimized for voice, does a poor job of carrying non-voice data. An all-digital system doesn't care whether the data being transmitted is voice, image or text. Only the endpoints are concerned with the real-world representation of the data. This is a much more flexible system. (An example: present repeater control systems use crude tone-signalling [DTMF] techniques to provide control capability. An all-digital system could send many times more control information in a much shorter time.)
- *Spectrum efficiency.* There are two trends of technology that hold great promise in the continued narrowing of digital voice bandwidth requirements. The first of these is the work being done in speech encoding (an example of one place where amateur systems can benefit from commercial technology). Speech encoding at speeds of around 16 kbit/s is a part of current commercial development efforts, and advanced vocoder designs are expected to produce acceptable speech quality at speeds as low as 4 kbit/s.[1][2] The second technological trend is that of digital signal processing (DSP). At present, the transmission speeds of mobile systems are limited to a few hundred kbit/s or so, a speed limitation that is carrier-frequency dependent.[3] The practical number of bits per baud is also limited to two or three.[4] DSP promises improvements in adaptive equalization that will improve those numbers, allowing us to squeeze more bits-per-second into a given bandwidth. (Note, though, that even present-day systems are perfectly workable, and can give bandwidths at least as narrow as present analog systems.)
- *Interference tolerance.* Although the FM capture effect provides a measure of tolerance to co-channel interference in analog systems, it is likely that properly designed digital systems will be even more tolerant, allowing greater frequency reuse. (Note that even the tolerance of FM analog is often not used in amateur systems. Systems without CTCSS access cannot tolerate any signal that will open the squelch.)
- *Time-division multiple access (TDMA).* As we will see shortly, TDMA has significant architectural advantages, and, while an analog TDMA system is possible, a digital TDMA system is far easier to implement.
- *Security.* A digital system can easily be designed to require an integral transmitter identification (call sign) in the data stream. This would provide traceability and debugging advantages.
- *Newness of technology.* One of the under-used facets of the amateur service is that experimental techniques can be tried out there without the need to produce a commercially viable system. Amateurs can build operational experience with new systems and, in so doing, add to the technology.
- *Flexibility.* Inherent in several of these points, but worth a mention of its own, is the ability of digital systems to change with the technology. For example, a system designed now would probably use speech coding rates on the order of 16 kbit/s. In a few years, when 8- or 4-kbit/s coding techniques are

accessible, existing digital systems could incorporate them without complete system redesign. In fact, such systems could be phased into service along with 16-kbit/s systems transparently to the user.

6. System Architectures

The architectural challenge is to devise systems that ensure that each station transmits in a manner that ensures reception by the intended recipient without unduly restricting the use of the system by other stations. Inasmuch as multiple access is required (see section 4), a key design decision will be to determine the best type of multiple access. Of the three classical techniques: frequency-division multiple access (FDMA), time-division multiple access (TDMA), and code-division multiple access (CDMA, or spread spectrum), we can select one or more or a combination of techniques.

FDMA is the system used in current cellular phone systems. In these systems, channels are segments of the band that are wide enough to contain the FM voice signal. They are no different from amateur repeaters in this respect. But in cellular systems, channels are assigned *on demand*. Such a controlled mechanism is a requirement in any useful multiple access system.[4] Cellular systems simply assign a particular frequency to the calling station for the duration of the call. This seems simple enough, and in an analog world it's by far the easiest approach. But it requires that the cell-site (or repeater) have multiple receivers and transmitters, an expensive proposition for amateurs. If full-duplex communications are required in an FDMA system, the fixed and mobile units must be capable of receiving and transmitting at the same time; a duplexer is normally required for this. However, in a digital FDMA system the transmitted bit rates are lower than in an equivalent TDMA system. This has advantages, not least of which is that bit periods can be relatively long compared to the delay-spread times.[5]

In a TDMA system, each station takes turns sending a burst of data on a particular frequency. This has the advantage of requiring only a single receiver and transmitter at the repeater, and allowing full-duplex communications *sans* duplexer. Also, by not fixing the channel bandwidth TDMA provides for more flexible selection of data rates: if a higher data rate is needed, the bursts of data are simply made longer. This latter characteristic may override other considerations.

CDMA, or spread spectrum has utterly different characteristics from either FDMA or TDMA. Spread spectrum operates by spreading the signal out over a large band of frequencies. One method of spread spectrum, *frequency hopping*, simply rapidly changes the frequency of the transmitter. The associated receiver "knows" the sequence of frequencies and follows along. Several such systems operating in the same band of frequencies can operate with little interference. In *direct sequence* spread spectrum, a high-speed (quite a bit higher than the data rate) pseudo-random bit stream is mixed with the signal to "spread" the transmitted spectrum. The receiver mixes what it receives with the same bit stream, resulting in a copy of the transmitted signal.[5] Adding stations to the band in such a system simply makes the apparent noise level increase, so interference degrades communication rather than interrupting it. CDMA has the advantage that the signal can be spread beyond the *coherence bandwidth*. This reduces the susceptibility of the system to multipath effects, which is particularly attractive for mobile applications. CDMA has great potential, if little operational history.[6] Amateur efforts have proved promising,[6] but it seems unlikely that CDMA will be the

---

[4] For a demonstration of why control is a requirement, listen to any CSMA simplex packet channel. 145.01 MHz will do in most areas.

[5] One of the technical challenges of spread spectrum is to keep the transmitter and receiver bit streams synchronized.

[6] The military has been using spread spectrum for years, but they aren't too forthcoming about the technical details.

technique of choice for next-generation amateur systems.[7]

7. Conclusion

Next-generation voice systems are a logical outgrowth of the push toward high-speed networks, so designing systems now to accommodate voice data is prudent. Such systems can address some of the spectrum-management issues facing amateurs today and can provide enhanced voice capabilities as well as integrated voice and data. The technical challenges of designing and implementing such systems will open up a world of interesting experimentation and building such as has not been seen in amateur radio in recent times. Some aspects of the possible approaches have been discussed here, hopefully in a way that encourages amateurs to begin serious consideration of next-generation voice systems.

References

[1] Chien, E. S. K., Goodman, D. J. and Russell Sr., J. E., "Cellular Access Digital Network (CADN): Wireless Access to Networks of the Future," *IEEE Communications Magazine*, June 1987, p 22

[2] Jayant, N. S., "High Quality Coding of Telephone Speech and Wideband Audio," *IEEE International Conference on Communications ICC '90*, IEEE, 1990, p 322.1.1

[3] Siew, C. H. and Goodman, D. J., "Packet Data Transmission Over Mobile Radio Channels," *IEEE Transactions on Vehicular Technology* May 1989, p 95

[4] Calhoun, George, *Digital Cellular Radio*, Norwood MA: Artech House, 1988, p 336

[5] Calhoun, p 366

[6] Kesteloot, André, N4ICK, "A Practical Direct-Sequence Spread-Spectrum UHF Link," *QST*, May 1989, p 14

[7] The author is willing to be convinced otherwise.

AVC_R_ISA: a MAC layer for NOS/net

**F. Davoli, A. Giordano (I1TD), A. Imovilli (IW1PVW),
C. Nobile (IW1QAP),G. Pederiva (IW1QAN), S. Zappatore (IW1PTR)
[al@dist.unige.it]**

**Department of Communications, Computer and System Science (DIST)
University of Genoa (IK1HLF)
Via Opera Pia, 11A - 16145 Genova, Italy**

ABSTRACT

In this paper we describe an implementation of the protocol AVC-R-ISA (Access Virtual Channel Radionet-Independent Stations Algorithm) performed in the NOS/net program. AVC-R-ISA may be a solution to the problem of traffic control in a packet switched network, like for example the amateur packet network. This multiple access strategy may be viewed as an alternative for the MAC layer to the popular CSMA and is aimed at improving the channel throughput with an optimized distribution of the channel itself among the users.

INTRODUCTION

Channel distribution among several users requires the application of access methodologies, of which various different kinds have been developed and successfully implemented. Amateur packet in one-level networks is well suited for a fully distributed access method like CSMA. When we consider a multilevel structure based on the presence of backbone stations and local area routers, the presence of a base station (a station involved in the data exchange at every turn-over) offers the possibility of enhancing the network throughput. This may be the case of a backbone station serving area routers being hidden to one another.

The access method called AVC-R-ISA (Access Virtual Channel Radionet-Independent Stations Algorithm) is based on an adaptive strategy, that tunes itself according to the network status. We briefly recall the adopted methodology: the backbone station, called master, well positioned to be received from all the other area routers, selects a group of them according to the R_ISA algorithm [1] and requests their transmission, being able to make an estimate of the traffic generated by every station known to be currently active on the network. As is consistent with intuition, enabling a set of stations with a "large" presence probability increases the possibility of a collision, whereas a set of stations with a "low" presence probability may more easily result in an empty transmission period. In both cases bandwidth would be wasted. There is a tradeoff where the number of enabled stations maximizes the network throughput, and the adaptive strategy tends to optimize it.

In the next section of this paper we briefly review the AVC_R_ISA principles and describe how the ideas presented in [2] have been improved.

In the last section we propose an implementation of AVC_R_ISA in the KA9Q Internet package. Our choiche is due to the large diffusion of this package that, together with the

availability of the well structured source code, allows a practical test of new ideas. The commands added to NOS/net are also described.

THEORETICAL ASPECTS

The theory of R_ISA shows that for an efficient traffic forecast, some parameters are necessary, including the identity of every station active on the network. In other words, the set of stations must be known before the network is activated.

AVC_R_ISA by-passes this restriction by including in its enabling list a set of "accessing stations ". For this group of stations, like for every other, a logical channel is defined, that simply consists in a single identifier assigned to every station at its entry into the network activity.

It is intuitive that only one common identifier (say, 0) may be assigned to all stations still out of the round, because they are "unknown" to the master. For an analysis of ISA and its evolution we refer the reader to [1,2]. The AVC (Access Virtual Channel) was added to R_ISA to allow stations enter the network. However, when several stations are ready to go on the air on the next access slot with the same identification, one must try to minimize the collision probability.

The easiest solution is to have these stations use a slotted-ALOHA access rule, whenever enabled to transmit by the ISA procedure. However, a fixed retransmission parameter for the ALOHA strategy, as previously adopted, turns out to be very inefficient. Therefore we have decided to use Rivest's pseudo-Bayesian broadcast [3] which is one of the most efficient strategies to optimize and stabilize slotted-ALOHA. It requires the estimation of the number of "backlogged" stations on channel 0, which is done by means of channel feedback, and must interwork with the estimation of presence probabilities performed by ISA. [4].

AVC_R_ISA frames cannot be received from a normal TNC running AX25 software, because bytes are added in front of the AX25 frame. AVC_R_ISA is located at a lower layer (MAC) than AX25, and is connection oriented. For this reason, single hop addressing is resolved at the AVC_R_ISA level and any AX25, IP or NETROM frame is simply encapsulated in it.

NOS/net NEWS

The modified NOS/net program is able to manage Master and Slave stations. The two types of stations have anyway available different frames. They have been slightly modified with respect to those proposed in [2], because during the experiments some more control frames showed to be necessary.

The Master can use the following frames:

1) Slave connection request; 2) Slave disconnection request; 3) Forced disconnection of all the Slaves; 4) Slave Synchronization; 5) Slave connection accepted; 6) Slave connection refused; 7) Slave disconnection accepted. The first 4 frames are activated by the master while the following 3 are sent in reply to Slaves' incoming frames.

The Slave can use:

1) Master connection request; 2) Master disconnection request. Information frames are then added to the above either for Master or Slave stations.

NOS/net transportability (PC, Macintosh, Amiga and Unix) has been maintained and also the set of services (AX25, IP, ICMP, TCP, UDP, ARP and, as applications, FTP, TELNET and SMTP). In the present version, the new MAC level has been implemented for the "asy" interface

(Standard PC Asynchronous Interface using the 8250 or 16550A), but we plan a version running other devices, like Software Packet Driver of the FTP Software Inc.

The software has been written preserving modularity. Data structures are included in a single data block, and they have been connected to the physical interface with the "attach" command. In the "iface" structure, two new fields are now available:
"isa_raw" and "isa_recv" (similar to the existing "raw" and "recv"), that activate the ISA channel control function.
After receiving frames from the upper levels, "isa_raw" generates a timed FIFO queue. In the case of channel congestion, to prevent the queue growing too large, a demon adjusts the time to live (ttl) of the frame, deleting it when the ttl reaches a threshold. This procedure is transparent to the frame structure, so that compatibility is maintained also with new upper levels and protocols.
The solution helps avoiding packet repetition on the channel. At heavy load conditions, the timeout of upper level managers may request a retransmission, even if the packet has not yet been transmitted for the first time or the transmission was successful but not yet acknowledged. In this case, removing "too old" frames from the queue prevents channel overload.
"isa_recv" is activated by the "network" demon after receiving an error-free ISA-frame. In this case, if the transmission queue is not empty, one frame is taken from the FIFO queue and placed in the tx_buffer, and then on the air after attaching the header containing MAC protocol information. The tx_buffer is then cleared only after the Master feedback on the successful reception of the frame is received.

If the program is running on the Master station, "isa_recv" calls the routine calculating the permissions table. This table will be transmitted to the slaves in a bit-map format. Obviously, Master and Slave procedures are sometimes different, but they may run together on the same node. A PC may act as gateway, being master of a network and at the same time slave of another network. Moreover, the master must provide sending on the air the sync for the network. This must be done also in absence of active Slave stations to enable the Access Channel: in this case, we prefer avoiding a continuous polling on the air, muting the Master and waiting for the presence of an ISA carrier. The Slave requesting entrance in the round on its part watches the channel, and after detecting no activity for a certain time, assumes the access channel is enabled and transmits.The Master must also pay attention to a station leaving the network without disconnecting from it. A procedure takes into account the elapsed time from the last transmission of every Slave and disconnects it when this time reaches a threshold. A "shutdown" frame is also available for the Master, that is able to signal a forced standby avoiding Slaves "hanging" on the channel.

A very important factor affecting the performance of the ISA algorithm is the geographical position of the master station. As already mentioned, two different versions of ISA/NOS were developed. The first one is devoted to end-users, i.e., it can only act as a slave station or as a slave gateway between ISA-net and other networks supported from the original KA9Q package.The second one is primarily devoted to area management, but it also supports the function of both master and slave roles, allowing the development of networks with hierarchical structure.

The hardware requirements for this latter version are:

- a 386 Personal Computer with a mathematical co-processor;
- a connection of its serial port to the CD signal detected from the TNC.

Of course, the ISA networking is possible only between stations running this modified version of the KA9Q package, as the frame structure cannot be decoded by the AX25-based available TNCs. In order to define the parameters affecting the ISA environment, we added some

new commands. The syntax is:

isa <subcommand> [<optional parameters>].

Their semantic is the same as used by P. Karn in the "Net User Reference Manual". Moreover, we modified the "attach asy" command, that now provides, in addition to the three previous operating modes (slip, ax25 and nrs), also the ISA mode, thus allowing to define an interface acting as master or slave station. Therefore the command syntax is the following:

attach asy <ioaddr> <vector> slip ax25 nrs isa <label> <bufsize> <mtu> <speed> [<flags>].

For instance, if one wants to attach an ISA interface on COM1, the command is:

attach asy 0x3f8 isa ax0 2048 256 9600.

Additional commands are the following:

ISA checkcount <iface> [<value>]

Sets or displays the time interval after which the Master disconnects an inactive slave station. This command is reserved to master stations and it activates only after no connection is present. The range (in seconds) of this parameter is from 10 to 64000.

ISA mastercall [<call>]

Sets or displays the ISA master station callsign. The used sintax is that of a standard AX.25 address. This command must be executed before any "attach" directive using ISA mode.

ISA <iface> [<n.station>]

Sets or displays the maximum number of slave stations that can be accepted by the master. This command is reserved to the master stations and it is accepted if no connections are present. The range of n.station is between 7 and 254.

ISA slotch0 [<value>]

Sets or displays the maximum number of transmissions after which the local identifier 0 (the access virtual channel) is certainly enabled. This parameter may be a critical factor for a good performance of the network. Therefore great attention must be paid in choosing this value, in order to avoid both a too long queue of slave stations waiting for connection and a too frequent enabling of the virtual channel access. This command is reserved to the master station and it is accepted only if no connection is present.

ISA status [<iface>]

Without argument, it displays the status of all inferfaces operating in isa mode. If <iface> is specified, only the related informations will be displayed.

ISA switch <iface> [on|off]

Sets or displasy the specified interface status. The transition from an inactive to an active status of a slave station is not immediate, but it needs the acknowledgement of the master station. If this command is executed by a master station, this sends a shutdown frame and stops the ISA activity.

ISA type <master|slave>

This command is available only for the version running on master stations. It allows a station to be master of one ISA network and slave of another in order to build a hierarchical structure.

CONCLUSIONS

We have considered an implementation of the AVC_R_ISA adaptive access strategy in NOS/net for operating in an amateur environment. The updated version of the program is now under test, and a network composed of a master station and six slaves will be soon on the air on the UHF band. The whole network is organized in a "master to area routers " configuration, the master ranging a radius of about 100 miles. Area routers will then collect the traffic of OM local stations spread over the North-West part of Italy offering a real test-bed for the system. We hope the utility of such strategy will be demonstrated, thus justifying the increased complexity.

REFERENCES

[1] M. Aicardi, F. Davoli, A. Giordano, "Radio_ISA_Net": a single-hop centralized packet radio network for PC-to-mainframe interconnection", IEEE J. Select. Areas Commun., vol. 7, pp. 219 - 225, Feb. 1989.

[2] F. Davoli, A. Giordano (I1TD), S, Zappatore (IW1PTR), "Local Distribution In The Amateur Radio Environment", ARRL 8th Computer Networking Conference, pp. 30 - 35, Oct. 1989.

[3] R. L. Rivest, "Network control by Bayesian broadcast", IEEE Trans. Inform. Theory, vol. IT-33, pp. 323-328, May 1987.

[4] F. Davoli, A. Giordano (I1TD), S. Zappatore (IW1PTR), "Architecture and protocol Design for a Car-to-Infrastructure Packet Radio Network", Globcom 90, to be published.

TITLE: A Built in TNC for the Toshiba Mod. T1000.

AUTHOR: Frederic de Bros, KXIS, July 26, 1990
22 Temple St.
Boston, MA 02114

1. INTRODUCTION:

The idea to have a TNC + laptop computer attached to a hand held transceiver seems attractive. Several home brew versions have been tried on SHARP PC 5000 and Tandy 100 computers. None were truly portable or clutter free.

We describe here how to affix a UMPAD (Universal packet assembler/disassembler) onto a modem board inside the Toshiba T1000 computer.

2. MATERIALS:

An asynchronous modem board type S231I (1) was purchased from Source One Systems (2). A UMPAD TNC with ribbon cable was purchased from PACCOM (3). A 27 pin right angle PC mount ribbon cable connector (part #SLEM 27R-2) was purchased from Burndy. A 27-strand non inverting ribbon cable was obtained from Electronic Innovations. A 5 pin Din female connector was obtained form Radio Shack.

3. METHODS:

The S331I was modified in the following way: The straight ribbon connector and the DB9DTE were unsoldered from the original board. The right angle 27 pin connector was soldered to the board. The 5 pin Din socket was mounted on the modem back panel with countersunk screws. The ribbon cable was replaced with a "non inverting" type. The 3 transistors on the modem board were bent down to a clearance of 5mm. The 0.1uf Tantalum capacitor was unsoldered and mounted horizontally. The cables leading to the battery connector in the T1000 were bent down, and backwards (towards the battery case). Two 5 mill. rubber standoffs were screw mounted in the original DB9 - Pin holes. The UMPAD was glued to these rubber standoffs. The UMPAD was connected to O, switched +9V, RxD and TxD (RS232) on the modem board. CTS and RTS were connected at the RS232 port of DCE and DTE respectively. The T1000 was modified by AXONIX (4) to a backlit ELT screen.

3. RESULTS

Figure 1 shows the modified modem board. Figure 2 shows the UMPAD with the radio/computer port. Figure 3 shows the assembled unit.

It works well in the car and in the office. Use is minimal though. We find it easier and cheaper to have a TNC + terminal at the various locations where packet can be used. Advantages are when going abroad, and to demonstrate/setup a system. We find that certain handheld transceivers are interfered by the clock (TH25AT e.g.)

4. FIGURES:

FIG. 1:
The S231I Modem board prepared for the UMPAD.

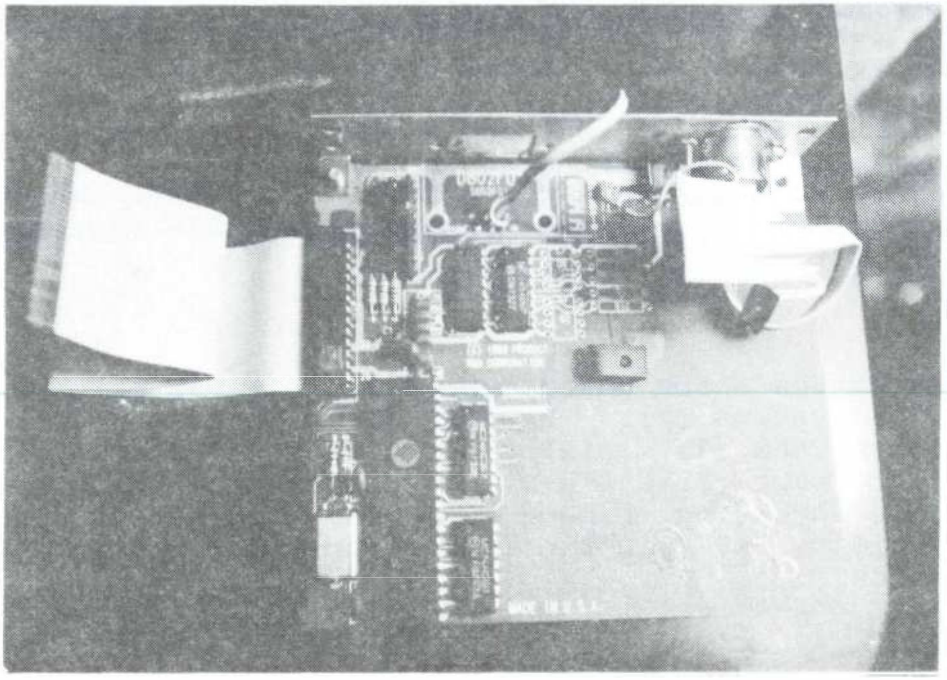


FIG. 2:
The UMPAD with the Radioport and computer interface.

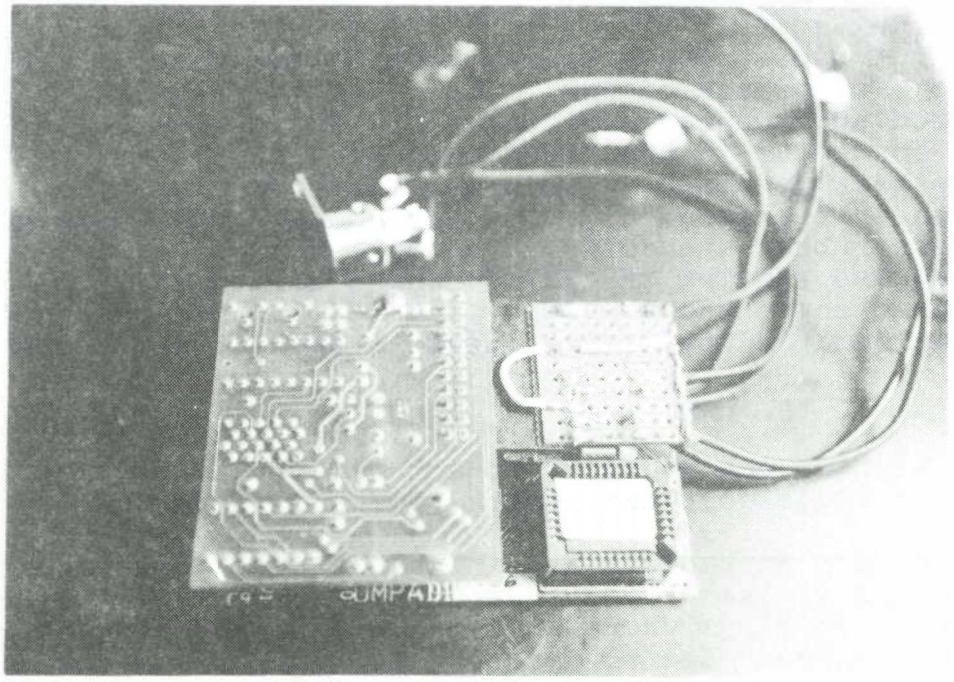
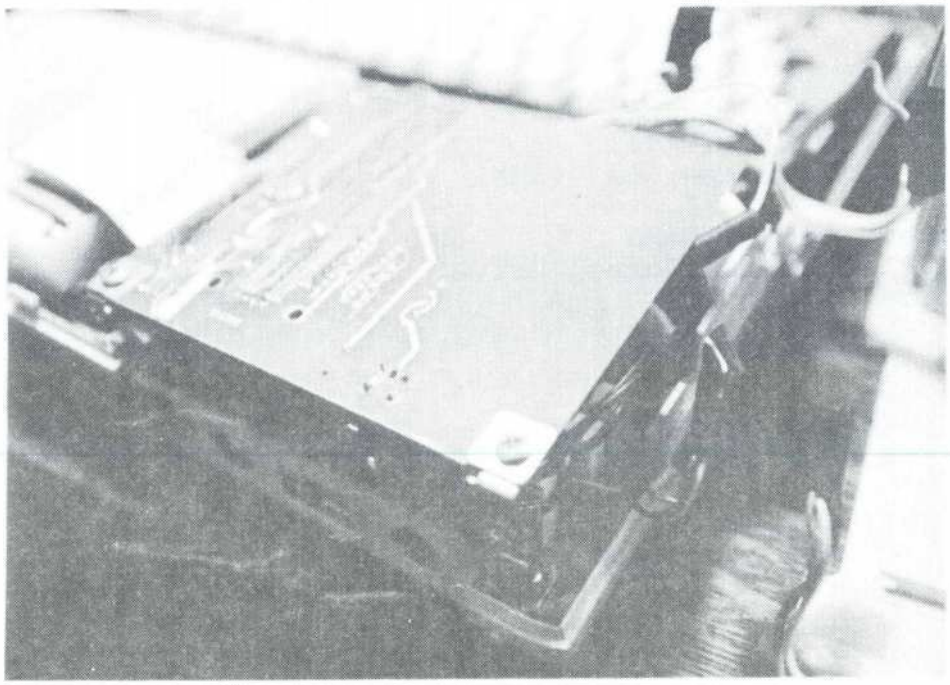


Fig. 3
The entire Unit mounted in the T1000.



5. DISCUSSION:

Several improvements are still possible. A better backpanel to house the DIN connector. The TNC should be software - switchable. The entire unit should be on a single board. Product R & D and PACCOM have been communicating about this project. An additional Hayes telephone modem should be installed on the board. RFI from the clock can be a limiting factor, if short antennae are used. Com 2 should be available as COM 3 on a DB 9 port. Use TTL levels rather than RS 232 to save even more battery power.

6. SUMMARY:

Portable PR with a T1000 computer is feasible with relatively little effort. A self contained unit attractively styled seems very desirable in the Amateur Radio Community.

7: REFERENCES:

(1) Product R & D Corporation
1194 Pacific Street
San Luis Obispo, CA 93401

(2) Source ONE Systems
13231 Champion Forest Drive
Suite 212
Houston, TX 77069

(3) PACCOM
3652 Wells Cypress Street
Tampa, FL 33607-4916

(4) AXONIX Corporation
2257 South 1100 East
Salt Lake City, UT 84106

A GPS DATA RECEIVER

By Dan Doberstein, MSEE
1406 Parkhurst Street
Simi Valley, CA 93065

This paper describes the construction of a Global Positioning System (GPS) single channel data receiver using the L1 carrier. A brief explanation of the GPS system is provided. The needed details of the GPS signal structure are also covered.

GPS uses Spread Spectrum techniques. It is one of the first world wide systems to implement this technology. These techniques will play an ever larger part in tomorrows communication systems.

The Global Positioning System

GPS is a satellite system that can provide users with time and position information. The satellites transmit satellite position and GPS time information to receivers on the ground or in the air. Orbits are half synchronous. There will be 18 satellites in the completed system. Nine satellites are up with the remainder to be installed within two years(?). Three satellites must be in view to obtain latitude and longitude (if altitude known). Four must be in view to obtain latitude, longitude and altitude. These two scenarios assume that the user receiver does not know GPS time. If position is known then GPS time can be obtained. When the system is complete this information will be available globally, twenty four hours a day.

GPS Signal Structure

The GPS signal uses two carriers L1 and L2. L1 is at 1575.42 MHz. L2 is at 1227.6 MHz. Both carriers have data information on them. All satellites use the same two frequencies. How does the user receiver know which satellite it is looking at? By use of a code that is impressed on both carriers that is unique to that satellite. Two codes are used; the C/A code and the P code. L1 has both the C/A and the P code impressed on it. L2 has either the P or the C/A code impressed on it. The C/A code is a 1023 pseudo random binary sequence. This code is open to public usage and is not "classified". The P code is an extremely long pseudo random binary sequence that is not open to the public and is "classified". For this reason the L2 signal is ignored. This paper addresses only the particulars of a L1 receiver. But the L1 signal has the P code on it too so how can you receive L1 with out knowing the P code? . It turns out that the method used for impressing the code(s)+data onto the carrier L1 allows for this provision.

Figure one is a simplified model of the satellite L1 generator. First note that all frequencies used are multiples of the 10.23 MHz reference. This is important and will be used later. We see from this diagram that the C/A code is at a 1.023 MHz rate while the P code is at a 10.23 MHz rate. The data clock is 50 Hz. It is derived from detecting the all ones state (10 1's in a row) of the C/A code, this occurs once every 1023 bits so this is at a 1 kHz frequency, and

dividing by 20. The two codes are exclusive OR'ed with the data to form the two BPSK modulating sources. The summation of these components results in a QPSK signal. Because of the 90 degree phase difference and the carrier spreading caused by the P code it is possible to build a C/A code BPSK receiver. In other words the P + DATA modulation on L1 can be ignored. That is what the receiver addressed in this paper does.

The minimum received power at the earths surface is -130 dBm for a 0 dB gain antenna. The satellite sends this out RH circular polarized. Due to the spreading of the C/A code the L1 signal is below the noise floor of the receiver. It can only be seen if the receiver has a copy of the C/A code to raise it "out of the mud" of the receiver noise. This process is called correlation and is at the heart of GPS receivers.

Figure 3 shows the spectrum of the L1 signal. The spectrum shows how the P code sin(x)/x envelope is below that of the C/A code sin(x)/x envelope.

The C/A Code

The C/A code modulation is far and away the most complicating aspect of this receiver. But without it the system would not work. So what does the C/A code do for us? First and foremost it allows "multiple access," or simply the ability for all the satellites to use the same frequency(s) to send data on. Without it a receiver would see a jumble of signals from all satellites in view. Antijam. The C/A code will reduce a offending CW signal that finds its way into the receiver by approximately 35 dB. The C/A code allows accurate determination of the distance between the satellites and the receiver. And as previously mentioned retrieves the L1 signal from below the noise floor. So although it may seem the C/A code adds needless complication it is vital to the operation of GPS receivers.

As noted above the C/A code repeats every 1023 bits. This repetition marker decoded from the all 1's state is called a epoch. The C/A code can be generated by two ten stage shift registers with appropriate tap points. Figure two shows the two shift register and the interconnections needed. The tap points on register G2 determine a unique C/A code for each satellite. Table 1 is a list of the tap points used for each satellite. Whenever a carrier is modulated by codes such as the C/A code this is referred to as "spreading" due to the resulting increase in bandwidth. The receivers job is to remove the C/A code (despread) so as to retrieve the data modulation. When a receiver has its code lined up with a satellites code and keeps it this way it is said to have code lock. When in code lock the receivers C/A code epochs are in sync with the transmitted C/A code epochs. A very important detail regarding the C/A code and for that matter the entire GPS system is that all satellites are synchronous. In other words the C/A code epochs are all lined up for all satellites at transmission time. This fact is used to make the position measurement.

Doppler

The velocity difference between the satellite and the user receiver causes a Doppler shift of the L1 carrier. This is on the

order of +7.0 kHz to -7.0 kHz. This causes some problems for the receiver. The problem stems from the low signal level of the received L1 signal. This forces a narrow IF bandwidth of approximately 1 kHz to achieve a tolerable SNR. To over come the Doppler the receiver must scan the possible Dopplers and then once the signal falls inside the 1 kHz IF filter it must keep it there using some sort of Doppler tracker. The Doppler is also present on the C/A code clock of 1.023 MHz. This is due to the coherent nature of the transmitter. The Doppler on the code clock is equal to the Doppler on the carrier divided by 1540, the multiplying factor between the 1575.42 MHz carrier and the code clock at 1.023 MHz. So for 7 kHz Doppler on the carrier we see approximately a 7 Hz Doppler on the code clock. Again the receiver must be able to tolerate this offset.

Data Modulation/Demodulation

Figure 1 shows that the GPS data rate is 50 bits per second. Ignoring the code modulation for the moment the data stream modulates the carrier via the BPSK modulator. Being that the content of the data stream is fairly complex, discussion of this will be postponed for now. Once the receiver removes the C/A code and has Doppler lock data demodulation can take place. Any type of BPSK demodulator can be used. I used one based on a PLL but many of the designs in the literature use the Costas loop demodulator.

Position Measurement

Although the receiver described in this paper does not make the position measurement some words on this are needed.

The key to the position measurement is the fact that all satellites are synchronous. This means that at all of the satellite transmitters the C/A code epochs are "lined up". This is indicated in figure 4. But to the user the epochs arrive at different times due to the different path lengths. Again see figure 4. These differences in path lengths are called pseudo ranges. It is the receivers job to measure these time of arrival differences w.r.t. his own clock and then compute the pseudo ranges and solve for user position and clock bias w.r.t. GPS time. This involves solving a system of four equations with four unknowns given the pseudo ranges and satellite positions from the measurements and satellite data. The equations are shown in figure 4.

The measurement of the path delays or pseudo ranges presents a problem for the single channel receiver such as the one described here. The single channel receiver must acquire and track four satellites sequentially as opposed to a four channel receiver which can track four satellites simultaneously.

If the user knows any one of the four unknowns an equation can be eliminated. The complexity of the sequential tracking and the solution of the equations requires a computer interface and a companion program. Also needed is a high speed digital latch to measure the path delays. At this time I have not pursued the position problem.

First Things First

Before anything can happen in the receiver whether its Doppler tracking, data demodulation or position measurement the receiver must line up its own generated C/A code with that of the transmitted L1 signal. If the two codes are not within two "chips" of alignment there is no signal for Doppler, data etc. A "chip" is the duration or length of one bit of code. This was for me the toughest problem to solve primarily because this is the one aspect of a GPS receiver that is completely different from conventional receivers. The method used to obtain code lock is the Tau-Dither circuit. This is a well documented method (but no actual circuits!) and is found in the references provided. I will leave the details of my implementation of this technique for later when the receiver hardware is discussed.

Why Build a GPS Data Only Receiver?

There are right now many commercially made GPS receivers on the market. I wanted to build my own receiver to 1) educate myself on receiver design, this is my first 2) Build a receiver that others could understand and if not duplicate have a good starting point 3) Be able to write this article that I hope will fill the void that currently exists in the literature on GPS and spread spectrum systems in general; a complete detailed description of an entire receiver. A benefit of this design is the fact that it is nearly all analog. Many of the commercial receivers digitize the IF and the signal disappears into a black box of digital signal processing. This is great from a cost/performance point of view but not from a intuition and learning point of view. The analog approach lends itself more to seeing the various and different trade offs in the design especially in the code tracking loop. And although the position problem is not addressed getting the data in my opinion is the toughest nut and has the most "fruits" to be gained. The data only receiver gives excellent insights that can be applied to other spread spectrum systems.

A L1 Data Receiver

Figure 5 shows a block diagram of the receiver. Some representative spectrums are shown at various points. The L1 signal is received using a quadrafilar circularly polarize antenna. The signal then enters a LNA. From here the signal goes through 60 feet of RG-214 (RG-213 could be used) to the crystal down converter. The down converter converts the spectrum at 1575.42 MHz to the 1st IF of 28.644 MHz. After passing through the 1st IF amp the signal comes to the Mixer/Correlator. This is just a DBM used in a slightly different way to achieve the 2nd IF of 5.7288 MHz. The only difference from conventional down conversion here is that the 2nd LO is phase modulated by the receivers own C/A code generator. This is where the C/A code is removed. If the receiver generated code is lined up with the code from the satellite then the 2nd IF will be present otherwise the receiver sees only noise. Assuming the codes are locked the 2nd IF spectrum is as shown. All the LO's for the 2nd, 3rd and 4th IFs are generated by division of the 114.613 MHz VCXO. The VCXO is needed here to compensate for Doppler and is controlled by the Doppler Scan/Track

circuitry. From the 2nd IF the signal goes through two more IF's to the final IF of 20 kHz. This low IF is chosen so that active BPF's can be used to achieve the narrow 1 kHz bandwidths needed. At this point the signal splits three ways into the blocks Tau-Dither code tracker, Doppler Scan/Track and Data demodulator. The Phase Modulation select switch is used for providing different signals to the phase modulator for the 2nd IF LO. All except the Code w/Dither position are used for testing only.

You might be wondering where the AGC is in this design. There isn't any. The receiver runs wide open. AGC is problem due to the C/A code spreading. To have AGC it would have to become active AFTER the L1 signal is correlated which means that a switching scheme would be needed. This is not that difficult but its probably better left to a computer to decide on AGC levels. The receiver works without it so why not?

A Word on IF Selection

The IF's as implemented in this design are less than optimum. They came about in sort of a evolutionary way. I had some oscillators that were close, the simulator I built made some choices for me etc. The choice of the 28 MHz 1st IF was based primarily on existing designs for the ham band at 1.3 GHz which used a 28MHz 1st IF. Also just the abundance of circuits in the Ham literature at 28MHz influenced me. The choice of the other IF's was based wrongly on my early attempts to make things multiples of the code clock. Not only is this not necessary you probably are asking for trouble doing this. Needless to say the design still worked and that is what counts!

Detailed Description of Receiver

The rest of this paper will be devoted to the details of the L1 receiver. I will start at the antenna and work through to the data demodulator.

The Antenna

The antenna is a quadrafilar type taken straight from the ARRL Antenna Book, 15th edition, pages 20-1 to 20-7. The design has good half hemisphere coverage which is exactly what is needed for a GPS receiver. The dimensions and other details of the antenna as built are given in figure 9. If you build this antenna make sure it is twisted the right way! This antenna has a gain of about 3 dB. It also is a fairly good match to 50 ohms.

The Preamp

Figure 6 shows a block diagram of the preamp. The first gain stage is a NEC Gasfet 32083. This FET has a gain of about 19 dB and a noise figure of 0.35 dB at 1575 MHz. The bias design of this fet was taken from The 1987 ARRL Handbook, pages 32-7. The input matching network is my own design. A lower cost FET such as the NEC NE720 or NE710 could be used here and not compromise performance. The element that must be tuned is the inductor in the input matching network.

Tuning consists of deforming the turns while observing gain and noise figure. Tuning for maximum gain will also work at the expense of a slightly higher noise figure, but still acceptable performance.

Following the GasFet gain stage is a double tuned BPF. The bandwidth of this filter is approximately 40 MHz. The two air variable capacitors are tuned for max gain at 1575 MHz. The small wires serve as a capacitor coupler between resonators. The insertion loss of this filter is about 2.5 dB.

After the BPF are two gain stages using the MCL MAR-8's. These RF gain blocks are easy to use and relatively cheap. There is no tuning associated with the these two stages. These two amps combine for about 30 dB of gain.

Figures 7 and 8 give the construction detail of the preamp. It is constructed on double sided G-10 board. It is one of the few places where I used printed circuit techniques. The entire circuit is mounted in a custom made aluminum box. A SMA connector is used for the antenna input and a N type for the output to the RG-214.

The overall performance of this preamp is +46 dB gain with a noise figure of 1.0 dB (1.6 dB if tuned by gain alone). This is acceptable but it could be better. The input match to the FET needs improvement as does the BPF.

Crystal Down Converter

The signal from the preamp is fed via RG-214 to the down converter. The cable has 6 dB of loss. In reference to figure 2 the RF input from the preamp first passes through a interdigital BPF to aid in image rejection. The detail of this filter is shown in figure 10. This design is a modification of the design in the 1987 ARRL Handbook, page 32-23. The filtered RF and LO signals are applied to a MCL DBM TFM-11 to convert to the first IF of 28.644 MHz. The IF is then passed to a tuned amplifier that has a gain of 55 dB and bandwidth of 3 MHz. The details of this amplifier are shown in figure 11.

LO generation consists of a 28.644 MHz xtal oscillator followed by a X54 multiplier chain to achieve the LO frequency of 1546.776 MHz. The LO has a power level of about 7 dBm. Figure 12 shows the two spectrums of the 1540 LO. One shows the harmonic content and the other close in purity. The doubler and the two triplers to 515 MHz were built dead bug style on a 4 by 8 inch piece of two sided G10 board. Each stage was enclosed in small shield box soldered to the board with individual +15 feed throughs to minimize coupling. The filters were also enclosed in small shield boxes.

Figure 13 shows the circuits of the doubler and the two triplers with detail on filter construction. Layout follows the schematic using dead bug construction techniques. The doubler design is taken from Solid State Design For The Radio amateur P. 44. The last tripler to 1540 MHz is done on a separate 2 by 6 piece of G10. The input is fed from the 515 tripler via RG-74. This allows the circuit path to be broken for separate tuning/testing. The 515 to 1540 tripler uses a MAR-8 for the X3, an interdigital BPF to select the third harmonic and a MAR-4 to bring the output to power up to approximately. 7 dBm. The printed circuit pattern in negative is shown in figure 15. Circuit layout and detail of the interdigital filter is shown in figure 16. The interdigital filter is a modification of the design used in the

1296 MHz transverter in the 1987 ARRL Handbook P. 32-17. The tips on construction and tuning apply but the dimensions given in figure 16 must be used. I used end covers on the filter box which are not used in the Handbook design. Figure 17B shows the xtal downconverter system.

A buffered 57.288 MHz output is picked off the 28.644 MHz doubler. This is used for testing but could also drive the dividers to generate the 2nd and 3rd LO's. This method would still need some sort of VCO for the 4th IF to allow for Doppler tracking.

If a 28.64 MHz VCXO is substituted for the fixed xtal oscillator the 4th LO could also be generated from this oscillator. This approach would eliminate the need for the additional oscillator and allow the 28.64 VCXO to handle Doppler tracking. The design used for the 10.23 code clock VCXO could be modified for this purpose.

28.644 MHz XTAL Oscillator

The circuit for the oscillator is shown in figure 17A along with layout. The crystal was obtained through ICM, Oklahoma City, OK. The oscillator could benefit from a temperature controlled oven as it can drift ± 20 HZ. This shows up as undesirable Doppler which eats away at the Doppler margin. Another undesirable is that this frequency is the same as the first IF. If any of the 28.644 makes it through the multiplier chain (and some does) it will appear in the 28.644 MHz IF. This is not as bad as it would first appear. The high gain of the preamp helps here as does the C/A code which as mentioned earlier will reduce any CW by 35 dB. A 57.288 MHz oven controlled overtone oscillator would solve both problems.

2nd IF/Correlator

The 28.644 MHz IF is fed to a DBM (MCL SBL-1) that does the correlation and down conversion to the second IF. The correlation is performed by phase modulating the LO drive. Phase modulation is done using a DBM (MCL SBL-1) driven at the I port by the C/A code. After the phase modulation the LO is BPF to limit the spectrum. As mentioned above when the receiver generated code is lined up within two chips the code on the L1 signal the second IF will be present, otherwise noise.

The 22.9 MHz LO is derived by dividing the 114.613 VCXO output by 10 and then filtering to get the second harmonic. This is done using the ECL counter MC10136. The details of the circuits for the 2nd, 3rd, 4th IF's and associated LO's is shown in figure 18.

The VCXO and the ECL dividers are mounted in a shield box. The ECL parts are super glued with legs flattened to G10 board. Connections should be short. The Pulse stretcher is used to lengthen the pulses from the MC10136. The rest of the 2nd IF generation circuitry is mounted in a small shield box. To drive the DBM phase modulator 7414 and 74128 are biased negative with respect to ground. This provides the bipolar drive signal necessary for BI-PHASE or BPSK modulation. The rest of the circuit is fairly straight forward and layout follows schematic with "dead bug" methods of construction on G10 board.

The MRF 901's used in both the 2nd and 3rd IF's are overkill and

2N2222A's or equivalent will work fine although the bias resistors may have to be changed.

3rd and 4th IF Generation

Nothing remarkable here just standard down conversion. The details are shown in figure 18. Construction is "dead bug" on G10 with the layout following the schematic. Each IF is mounted in its own shield box with SMA connectors for the inputs and outputs. The 4th IF box does not have the 20 kHz BPF inside. This is done at the C/A code Scan/Track and Doppler Scan/Track board.

114.6130 MHz VCXO

This is the only oscillator I did not build. I used a Vectron oscillator that had a 100 kHz FM range. The frequency stability requirements of this oscillator are mild due to the fact that it is controlled by the Doppler scan/track circuit. The frequency drift should be under 400 Hz. Its the "pull" range that is hard to come by. The oscillator used should be able to be pulled at least 40 kHz at 114 MHz. This oscillator can be replaced by making the 28.644 xtal oscillator a VCXO. See above under crystal downconverter.

Operation of the Tau-Dither C/A Code Scan/Track System

Figure 19 shows the block diagram of this system. The purpose of this system is to search for C/A code alignment and once found keep it there. Both functions are implemented by controlling the code clock frequency via the VCXO. The code clock VCXO is driven either by a constant voltage for Scan or a varying voltage for Track. An analog switch chooses between the two. When the switch is in Track position the system is said to be "closed loop". When in the Scan position the system is "open loop". Various waveforms and spectrums are shown accompanying the block diagram of figure 19. Some waveforms are open loop others are close loop. The open loop waveforms are marked with an asterisk.

<u>Open Loop Operation</u>:

In order to understand closed loop operation it is first necessary to understand open loop operation. In the open loop mode the code clock VCXO is held at a constant frequency offset from 1.023 MHz, the zero Doppler code rate. This frequency difference, about 10 Hz, causes the receiver generated code to "slip" by the code on the L1 signal. The time it takes for the two 1023 bit codes to make a complete pass is N/DF seconds; where N=1023 and DF=10Hz the frequency difference between the two code clocks.

Ignoring the effect of the Dither for now lets look at what occurs as the two codes slide by each other. As the two codes slip they will come to point where they "correlate". Correlation is when the two codes are within two chips of alignment. As correlation occurs the 20 kHz IF will start to appear at the output of the BPF. Before correlation the output of the BPF is just noise. The 20 kHz IF doesn't appear all at once, rather it builds in amplitude reaching a peak when

the two codes are in perfect alignment. As the two codes continue to slip the 20 kHz IF amplitude decreases until we are back to our noise output from the BPF. This process gives rise to the characteristic triangular shaped correlation pulse at the output of the Full Wave Detector. The width of this pulse is given by 2/DF, where DF is defined as above.

Now lets look at the effect of the Tau-Dither. Tau dither is used to generate a voltage that can be applied to the code clock VCXO as to keep the codes in "lock". It does this by determining which side of the correlation pulse the receiver generated code is on and how far it is from the peak point. If we know which side of the correlation pulse we are on we can determine if the receiver generated code should be advanced or retarded with respect to the received code. Knowing how far off we are tells us how far we need to move the code.

This information is generated by "dithering," or switching, between two versions of the receiver generated code. One version is delayed the other is not. The delay used here is about 1/2 microsecond or 1/2 chip. As the two codes slip through correlation the dither switching induces AM on the 20 kHz IF. The frequency of the AM is the same as dither clock frequency. The amplitude of this AM increases to a maximum and then decreases to zero when the codes are in alignment. As the two codes continue to slip it again grows and diminishes to zero as we pass the correlation point. This "double hump" waveform can be generated by detecting the output of the dither BPF. This is shown in the block diagram but it is not used or needed in the actual circuit. At the midpoint of the double hump, at code alignment, the induced AM goes through a 180 degree phase shift. The phase shift contains the advance/retard information while the amplitude of the AM contains the "how far" information. The AM and its 180 degree phase shift at code alignment is caused by the triangular shape of the correlation pulse.

The dither induced AM is "picked off" the full wave detected 20 kHz IF with a BPF tuned to the dithering frequency. The dither AM is now multiplied by the dither clock reference. This recovers both the phase and amplitude information simultaneously. The output of the multiplier is lowpass filtered to give the "discriminator" error output. The 180 degree phase change causes a polarity reversal which produces the "S" shape discriminator output. We now have voltage whose polarity tells us whether to advance or retard our code and whose amplitude tells how much.

The low pass filter after the multiplier serves the same purpose as the loop filter in the more familiar phase locked loop. Much of the analysis of the code loop can be done using the tools from phase locked loops with slight modifications.

Closed Loop Operation:

When correlation occurs the carrier detector senses the presence of the 20 kHz IF and flips the switch from Scan to Track. The code clock VCXO is now being controlled by the error voltage from the discriminator. The discriminator voltage constantly "pushes" the receiver code in the proper direction so as to keep the two codes in lock. When the discriminator output is positive the code should be advanced, or lower the VCXO frequency. Just the opposite for negative

voltages. In this manner the code clock VCXO is frequency modulated to keep the codes in alignment.

Getting a feel for the code scan/track process is best done with the aid of L1 simulator. With the simulator the L1 code and the receiver code can be displayed on a two channel scope. By triggering on either the L1 or receivers code epoch that code can be made to "stand still" on the scope. The other code will move across the screen in the direction determined by the sign of the frequency difference and at the rate determined by the magnitude of the frequency difference. If the loop is held open all the open loop waveforms can be observed as the two codes slide through correlation.

If we allow the scan/track switch to operate we can observe the transient and steady state behavior of the code track process. In steady state the tracking jitter can be measured.

Seeing the movement and dynamic properties of the scan/track system on a scope is hard to beat for getting a handle on this tricky problem.

C/A Code Scan/Track Circuit

Figure 20 shows the schematic diagram of the code scan/track circuit. This is a direct implementation of the block diagram of figure 19 with the exception of the dither detection which is omitted as stated above.

20 kHz IF Detector:

This is a full wave detector. LF 356 op amps are used for there high slew rate. This circuit was taken from P. 241 of the OP Amp Cookbook by Walter Jung, 3rd ed.

Carrier Detector:

The Carrier Detector consists of a LPF/Threshold detector, 10 turn pot, retriggerable one-shot, an OR gate and a LED. The one shot/OR gate are used to debounce the threshold detector output by extending the time that the analog switch is set to "track". The LED is lit for a carrier level above threshold. Threshold is set by the 10 turn pot.

Dither Bandpass Filter:

A single op amp active filter taken from P. 154 of the Active Filter Cookbook by Don Lancaster. This circuit has a Q of about 5. Because of the low frequency of the dither signal 741 type op amps work fine here. The filter is tuned to about 200 Hz. See below.

Four Quadrant Multiplier/LPF:

I used the Exar 2208 multiplier though any four quadrant multiplier would work due to the low frequencies involved. The details of operation and proper connections can be found in the 1988 Exar data book. A simple RC network at the output of the multiplier serves as the loop filter. The values of R and C will greatly effect code

tracking performance.

Analog Switch/Amplifier:

I used a MC14051 CMOS analog switch for the scan/track selection operation. Other switches could be used, even relays, with proper circuit modifications. This is a one-of-eight mix that is used as a SPDT switch. The other inputs are grounded. This switch can only handle plus or minus 5 volt signals. Because of this a 741 gain stage is added after the switch. The switch may be damaged if inputs exceed plus or minus 5 volts.

C/A Code Tau-Dither Loop Trade-Offs

The C/A code loop is a feedback loop and like all such loops there are trade offs in the design. The selection of the dither clock frequency is bounded by the IF bandwidth and by the need to "measure" the code error as frequently as possible. The maximum scan rate is limited primarily by CNR which in turn is related to IF bandwidth. The remaining "jitter" on the code lock is affected by the CNR, the loop filter, IF bandwidth, dither clock frequency and dither delay.

Using a Tau-Dither type of code tracking loop reduces the correlated IF amplitude by about 1.5 dB for this design. This loss gets larger for longer dither delays. Longer delays give better acquisition performance but result in more jitter.

Consequently the C/A code loop design is a trade off between the various requirements that are at odds with one another. The bibliography section lists a number of references on this topic.

Adjusting the C/A Code Scan/Track Circuit

There are a number of adjustments that must be done to the C/A Scan/Track loop for proper operation. The adjusting is done via ten turn pots. The schematic, figure 20, shows the location of each pot with the exception of the Dither frequency pot. With the exception of the Threshold pot all are of the miniature, screw type adjustment variety.

Dither Frequency Adjustment:

The dither BPF is not tuned but instead the dither frequency is changed to fall into the fixed pass band of the dither BPF. The dither clock is a 555 timer with a pot for frequency adjustment. The dither clock is part of the C/A code generating circuit and is shown in figure 25.

The tuning procedure is as follows: a simulated 20 kHz carrier is AM modulated with the dither clock. This could be done with a 555 timer and an AND gate. The output of the 200 Hz dither BPF and the 200 Hz dither clock are displayed on a two channel scope. Change the dither clock frequency until the two waveforms are either in phase, 0 degrees, or 180 degrees out of phase. This does two things; 1) It puts the dither clock frequency in the center of the Dither BPF 2) It aligns the PHASE of the detected dither AM with that of the dither clock. This is important. It insures that the multiplication of the

dither AM and the dither clock is done properly. The 0 or 180 degree alignment depends on where the dither clock is picked off; either before or after a logic inversion. Figure 22 shows the dither clock tuning setup. This brings up another point. If you are having trouble "locking" try inverting the dither clock reference to the multiplier. This will invert the error voltage polarity so that advance is now a negative error voltage and retard is now a positive voltage. Or visa versa.

Code Clock Bias Adjustment:

The code clock VCXO only responds to positive voltages. Therefore a bias is needed to allow for bipolar drive. This is provided by summing a bias voltage with the error voltage from the multiplier. The bias voltage is set to approximately 6 V dc. This assumes the code clock VCXO is adjusted properly. To adjust the bias a good frequency counter is needed. It should be accurate to ± 5 Hz at 10.23 MHz. The MULTIPLIER OFFSET adjustment should be done before this adjustment.

The adjustment is done as follows; disconnect the dither reference signal and ground pin 3 on the XR2208 multiplier. This should force the output of the multiplier to zero volts. Ground the IF input also just to be on the safe side. Force the track mode by reducing the threshold adjust pot until the carrier detect LED lights. This connects the VCXO control point to the output of the bias summer. Now adjust the bias pot until the frequency of the code clock is 10.230000 MHz.

Multiplier Offset Adjust:

This adjustment compensates for any d.c. offsets in the multiplier. Ground the multiplier inputs as detailed in the Code clock bias adjustment. Now adjust the multiplier offset pot until the voltage on pin 11 of the XR2208 multiplier is zero volts.

Threshold Adjust:

Threshold adjustment is provided by a ten turn precision pot with a vernier scale. Adjustment consists of turning the pot until the carrier detect LED just starts to flicker and then backing off a little so that the LED goes out. This must be done with the entire system up and consequently is a measure of the noise floor. This sets the level at which the system will declare that a carrier is present and correlation is assumed. The vernier scale is important. It enables resetting the threshold to various levels and monitoring any increase/decrease in the noise of the system.

20 kHz BPF adjustment:

The 20 kHz IF filter is shown in figure 23. There are three identical active BPF's; Two are used for the Doppler tracker and one is for the 20 kHz IF. The center frequency of these filters is adjusted by a 10 turn pot. The adjustment of the IF filter is done at the same time as the Doppler filters. The details of the IF tuning are covered in the Doppler Scan/Track section.

C/A Code Generator

Figure 25 shows the schematic of the C/A code generator. This is a single code version of the generator shown in figure 2. Six 7495 four bit shift registers are used to implement the two 10 bit shift registers shown in figure 2. It is hardwired for satellite vehicle 9, tap points 3 and 10. With the addition of some exclusive or gates and switching logic other codes could be generated.

Two versions of the code are generated. One delayed, one not delayed. The delayed code is the result of feeding the on time code through eight 74L04 inverter gates. This is about a 1/2 microsecond delay or 1/2 of a chip. These two versions are switched back and forth to produce the dither code. The dither clock is a 555 timer followed by a divide by 2 for square wave output. The dither frequency adjust pot is set as described above.

In addition to the dither code three other waveforms can be used for phase modulation of the 2nd LO. The three are code clock, on time code and CW (no phase mod.). These are use for testing purposes only. The 74153 is used to select one of these modulations by the setting of S1 and S2, the phase modulation select switches.

Some method must be used to reset the generator to the all ones state. This is done by the code reset button. After being debouced by the two AND gates it is synchronized to the code clock by the D flip-flop 7474. The input to the shift register is held high by this synced pulse. This loads the register with all ones.

The C/A all ones state or epoch is generated by AND'ing all ten output bits of the 10 bit shift register. Two 4 wide 7421 AND gates and two 2 wide 7408 AND gates are used for this function. The output is a 1 kHz pulse train.

C/A Code Clock VCXO

The C/A code clock is a 10.23 MHz VCXO followed by a divide by ten to produce the 1.023 MHz code clock. Figure 26 shows the schematic of the code clock generator. The VCXO is ovenized to reduce frequency drift. This oscillator will drift less than 2 Hz at 10.23 MHz. VCXO pull is about 250 Hz. This range can be increased by increasing the cap in series with the MV 209 diode.

Following the oscillator is a low pass filter used to reduce the upper harmonics. A MAR 3 is used as a buffer amp and a pick off point is provided for frequency monitoring. A 74160 is used to do the divide by ten operation and the 1.023 MHz is sent out via the 7404 inverter.

The oscillator/filter/buffer were built dead bug style on a 1 1/2 X 3 inch piece of G10. This was fitted with a tin cover. The heating resistors were epoxyed to the cover. The completed oscillator was then put in a tight fitting styrofoam box. The power, heater and 10.23 MHz outputs were brought out of the box. RG-74 was used for the 10.23 MHz outputs.

Heater control is provided by comparing the voltage from the LM336, a precision 2.5 volt reference, and a LM335 temperature sensor. When the divided voltage from the LM335 is below 2.5 volts the heater is turned on. When above its off. The heater is made of four 220 ohm, 1/2 watt resistors in parallel. The set temperature is controlled by the 10k ten turn pot.

Code Clock VCXO Frequency Adjustment:

Frequency can be adjusted by the control point to the tuning diode, the piston trimmer on the oscillator board and to a lesser extent the oven temperature via the ten turn pot. A good frequency counter is needed to adjust this oscillator, accurate to within a few Hz at 10.23 MHz.

The heater must be adjusted first. Prior to applying power to the heater control adjust the Temp. Set Pot for lowest temperature, i.e., wiper at the top. Now apply power and slowly bring up temperature to about 125 F or about 3.2 V dc at the temp sensor. Once the temperature has stabilized adjust the frequency of the oscillator about 100 Hz BELOW 10.23 MHz using the piston trimmer. It is set low to allow for the d.c. bias added to the code error signal. If the trimmer cannot bring it into tune try changing the oven temp a little and readjust piston trimmer .

If a higher code scan rate is desired set the oscillator even lower. The limit here is the total pull at the control point which as mentioned above is about 250 Hz. With this range 180 Hz below should be feasible and would result in about the highest scan rate that could be used with this system.

The design for the oscillator was inspired by designs found in the 1988 ARRL Handbook. The heater control circuit came from the data sheet applications for the LM 335 sensor in the Linear Data Book from National Semiconductor.

Doppler Scan/Track System

As discussed above the narrow IF bandwidth of 1kHz and the ± 7 kHz Doppler on the L1 carrier make it necessary to do a Doppler scan and track operation. Figure 23 shows the Doppler scan/track sub-system.

Scan:

The scan function is implemented by letting the 8 bit UP/DOWN counter free run with the UP/DN input held low. The counter is driven by a 1.5 sec period clock. The output of the counter is converted to analog by the DAC. This results in a sawtooth waveform with a 6 minute period coming out of the DAC. The output of the DAC is summed with a bias to account for zero Doppler. This is fed to the 114.613 MHz VCXO. In other words the L1 signal is swept through all possible Doppler offsets by the VCXO until it "falls" into the Doppler filters. The range of the Doppler scan is determined by the voltage divider at the DAC output amp. The range as shown is about ± 5 kHz.

All three filters will have some of the signal in them since they are about 500 Hz apart and have approximately 1 kHz bandwidths. When this condition occurs in conjunction with correlation the carrier detect circuit is tripped. This activates both code and Doppler track circuits.

Track:

The output of the two Doppler filters is detected and low pass

filtered. The comparator subtracts these two levels to determine which filter has more of the L1 signal in it. In the track mode control of the Up/Down input of the 8 bit counter is switched to the output of the comparator. If the L1 signal is more in the lower Doppler filter the comparator tells the counter to count up. If the L1 signal is more in the upper Doppler filter the comparator tells the counter to count down. The output of the counter, when converted to analog by the DAC, keeps the L1 signal centered in the 20 kHz IF filter.

The two D type flip-flops are used to sync the UP/DN and Carrier Detect signals to the 2 second clock pulses. The UP/DN LED gives a visual indication of which direction the counter is going. When tracking Doppler the UP/DN LED should toggle between high and low at about the 2 second clock rate. I used a Burr Brown 8 bit DAC but any 8 bit DAC should work.

Adjusting the Doppler Scan/Track Circuit

There are only two adjustments to the Doppler scan/track system. Doppler zero and the Doppler filters center frequency.

Doppler filter adjustment:

Figure 24 shows the schematic of the active filter used for the three Doppler filters. The filter is tuned via R1 a ten turn miniature pot. A signal generator or 555 timer is needed as an input for tuning the filters. The lower Doppler filter is tuned to approximately 19.5 kHz. The upper Doppler filter is tuned to approximately 20.5 kHz.

The middle Doppler filter, which is used for providing the 20 kHz IF to the code loop and data demodulator, is tuned by one of two methods. The first method uses a signal generator to find the frequency at which the comparator toggles. This is the frequency that the middle filter should be tuned to. This method will produce adequate results.

The second method uses a either a 1575.42 MHz RF or 28.644 MHz IF simulator. The middle filter should first be tuned by the first method so this really is the "fine" tuning part. Assuming a simulator is available connect a -40 dBm 28.644 MHz CW signal to the 1st IF input or a -120 dBm 1575 MHz CW signal to the preamp input. Set the phase modulation select switches to CW. This allows the CW to pass "unspread" through the system and no code lock is needed. Allow the system to track the CW signal. Using a voltmeter or scope tune the middle filter for maximum voltage or amplitude. This method closes the loop and should produce a more accurate result.

Doppler Zero:

The VCXO may not be set exactly to the frequency that would put the L1 signal in the center of the 20 kHz IF filter when L1 has zero Doppler on it. A bias voltage is summed with the DAC output to compensate for this error. The VCXO does not have to be set closer than ± 200 Hz.

There are many ways to adjust the VCXO for zero Doppler. The most straight forward is to use a frequency counter to measure the VCXO frequency and to adjust the bias pot accordingly. If there is no

offset in the oscillator used in the downconverter then the VCXO frequency for zero Doppler is 114.6130 MHz. The voltage from the DAC must set to zero to do this adjustment. The switch on the plus input of the bias amp is used for this purpose.

Acquisition Times

The combined requirements of scanning the C/A code and Doppler lead to some questions about how long it takes acquire the L1 signal. Assuming that the scan rates and bandwidths remain constant CNR is the determining factor. Larger CNR's result in shorter acquisition times while smaller CNR's lead to longer ones. For a CNR of about 20 dB this system will have a average acquisition time of 5 minutes. This time could be decreased by increasing the scan rates. The code scan rate would have to be increased before the Doppler scan rate can be increased. The scan rates used here are slow compared to what could be used.

Data Demodulator

A block diagram for the data demodulator is shown in figure 27. The Schematic is shown in figure 28. This design is based on a block diagram found on P. 211 of Phase Locked Loops by Roland Best.

After the C/A code is removed a 50 BPS data stream remains. This is present on the 20 kHz IF as BPSK modulation. The first step in the demodulation of this data is to hard limit the 20 kHz carrier. This removes any amplitude variation. The limited IF is now fed to a CD4046 PLL chip. The PLL responds to PHASE changes on the data with an impulse, not a pulse. The impulses occur at every phase transition. Since there are two phase transitions per data bit, one for the rising edge and one for the falling edge, there are two impulses per data bit.

The impulses can have either positive or negative polarity and can flip arbitrarily. To make all impulses have a positive polarity a full-wave rectifier is used. This is done with a couple of LF356 opamps and four diodes. This design is lifted from P. 241 of the OP Amp Cookbook by Walter Jung, 3rd edition. After rectification a non-inverting gain of 3 is used to increase the impulse amplitude.

Now a 7414 is used to clean up the impulses and bring them to TTL levels. A oneshot with a output pulse width of 8 milliseconds is used to stretch the impulse into a pulse and add some noise immunity.

The data pulse train is now divided by two in order to eliminate the two pulses per bit and to "square it up". This is done with a 7474 D-type flip-flop. The Q output is taken as the demodulated data stream. Now that we have the data signal we still need to know when to sample the data waveform. The data clock is needed.

The data clock can be reconstituted by dividing the 1 kHz C/A code epochs by 20. This gives the needed 50 Hz data clock. This is done using a 74160 decade counter for the divide by ten and a 7474 for the divide by 2 to get the divide by twenty. This circuit is shown in figure 28.

The only adjustment in the data demodulator is the PLL VCO frequency. The free running frequency of the VCO should be 20 kHz, the IF frequency. Adjustment can be done when tracking a satellite. When

the Doppler tracker settles down and is tracking the satellite monitor pin 9 of the PLL on a scope. Now adjust the VCO frequency via the 5 kohm pot until the D.C. level on pin 9 of the PLL is about zero volts. This should insure that the free running frequency of the VCO is close to 20 kHz.

Another method is to use a signal generator to generate the 20 kHz IF. This is fed to the Doppler filters and the frequency varied to find the point at which the Doppler comparator toggles. Now adjust the PLL VCO as above.

The performance of the data demodulator could be better. Signals with CNRs below about 15 dB will produce bit errors on the order of 1 in 600 or worse. Above this the performance is quite acceptable and good results were obtained. The detected IF level that I used as a threshold was about 2 volts.

Results

The receiver was used to track and receive data from satellite vehicle 9. The data was taken November 23, 1989 at approximately 3:30 AM PST. The satellite passed nearly overhead at this time. this insured good signal strength. Figure 29 shows a elevation plot of satellite vehicle 9 for Nov. 23 1989 at my latitude and longitude. The afternoon occurrence was at a low elevation angle and did not provide very good data but the receiver was still able to track it.

Figure 30 shows a recording of detected IF level and VCXO control voltage from the Doppler tracker for the early morning occurrence. Figure 31 shows the received data spectrum from the satellite on the 20 kHz IF taken at the afternoon occurrence. The Doppler plot shows the sawtooth scan until the satellite is "locked on" by the receiver. The detected IF slowly rises as the satellite comes up. Zero Doppler is a little off due to some frequency drift in the downconverter. As the satellite goes back down the Doppler increases to the point where lock is broken. This is due to the drift and the fact that I was only set up for about a ± 5 kHz range of Doppler. Given more Doppler range the receiver should have tracked for about another hour. Once lock is broke the Doppler goes back to the scan mode and the Det. IF shows just the noise level. There is a brief "lock up" after the loss of lock but it is short lived.

The Data

The data was read using a HP3270 computer that had an eight bit input port. The clock was attached to bit zero and the data to bit 1. The data line was read on either the rising or trailing edge since it was not certain at the time which would give the best results. It turned out that the trailing edge data was the good stuff.

Figure 32 is a sample of the data. The data is sent in 30 bit words. Ten words make a subframe. There are five subframes for a full frame of data. The first 8 bits of each subframe are the same. This is the preamble. After the data was read into the computer a program was used to "hunt" for the preamble pattern. When five preambles are found that are 300 bits apart a "sync" is declared.

When the "sync" is found the program then sorts the data into 30 bit words and 10 word subframes. The subframe ID's are contained in

bits 20-22 of the second word in each subframe. The subframe ID's are printed in a column at the right margin. Both the inverted and non-inverted are printed as the program did not check for data polarity which can be arbitrary. The data polarity can be determined from the preamble. The polarity of the data shown is correct and the left hand column of subframe ID's is the correct one. Of course one could also tell by the sequence, i.e. 1, 2, 3, 4 and 5 etc, as they are sent in order. The right hand column of numbers is a count of the data bits as they where read into the computer.

The important item here is the preamble. When you see that pattern repeat every 300 bits you know you have the data! Figures 33 and 34A/B show some the internal detail of the data format. These figures were taken from the ICD-GPS-200 document.

This is as far as I have taken the data analysis. To go further requires a lot more programming and possibly computer control of the receiver.

Obtaining the ICD-GPS-200 Document

This key document is available from the GPS program office. A letter must be sent to the address given below stating name and purpose. Unfortunately they do not have to send you a copy! Although the document IS NOT classified the program office will decide on a case by case basis if the requesting party can be "trusted" with a copy. I could get no firm commitment on the qualifications needed to get the document, although I was told that if my address was in Moscow I would not receive a copy. Strangely, they did not object to the inclusion of the C/A Code Tap Point table and the Data Frame Format figures, the very information I found to be the most difficult to obtain!

The C/A code portion of the L1 signal is for public use. The GPS system was paid for and is owned by the people of the United States. Currently there are foreign and domestic manufacturers who are profiting from C/A code receivers. If the large corporations have access to system information shouldn't the experimenter and the entrepreneur?

GPS Program Office address and phone number:

Space Systems Division/MZEE
ATTN: Data Manager
Los Angeles Air Force Base
P.O. 92960, Los Angeles, CA
90009-2960 Tel 213-643-0880

Acknowledgements

I would not have been able to build this receiver without the many fine publications provided by the HAM radio people. In particular the ARRL publications. They had examples of real circuits that worked. I would also like to thank Todd Carstenson who wrote the data capture program and provided a source of encouragement. There are many people who added bits and pieces to this endeavor, librarians, co-workers, people in industry, etc, who I would also like to say thanks.

BIBLIOGRAPHY

Ham Publications:

The 1987 ARRL Handbook, 64th Edition.

The ARRL Antenna Book, 15th Edition.

The ARRL Electronics Data Book, 2nd Edition.

Solid State Design for the Radio Amateur, Wes Hayward and Doug DeMaw.

QRP Notebook, Doug DeMaw, W1FB, ARRL Pub.

The Satellite Experimenter's Handbook, Martin Davidoff, ARRL Pub.

Proceedings of the 22nd Conference of the Central States VHF Society, 1988, ARRL Pub.

VHF/UHF Manual, 4th Edition, G.R. Jessop, G6JP. Published by the Radio Society of Great Britain (RSGB).

Papers on GPS:

ICD-GPS-200, Interface Control Document, Rockwell International Corp.

GPS Signal Structure and Performance Characteristics, J.J. Spliker, Jr. Journal of the Institute of Navigation, Vol. 25, No. 2, Summer 1978, P. 121-146.

A Marine Navstar GPS Receiver, Reuben E. Maine, Global Positioning System Papers, Institute of Navigation Vol. II, P. 36-43.

A Low Cost Receiver for Land Navigation, Kai P. Yiu, Richard Crawford and Ralph Eschenbach, Global Positioning System Papers, Institute of Navigation Vol. II, P. 44-60.

All Digital GPS Receiver Mechanization, Peter Ould and Robert J. Van Wechel, Global Positioning System Papers, Institute of Navigation Vol. II, P. 25-35.

Performance/Cost Ratio Optimized for GPS Receiver Design, Ralph Eschenbach and Roger Helkey, Microwave System News (MSN), Nov. 1984, P. 43-52.

Navstar Global Positioning System Offers Unprecedented Navigational Accuracy, Robert P. Denaro, MSN, NOV. 1984, P. 54-78.

Code Tracking Papers:

The Delay Lock Discriminator - An Optimum Tracking Device, J.J. Spilker, Jr. and D.T. Magill, Proceedings of the IRE, September 1961, P. 1403-1416.

Delay - Lock Tracking of Binary Signals, J.J. Spliker, Jr., IEEE Transactions on Space Electronics and Telemetry, March 1963, P. 1-8.

Application of Delay - Lock Radar Techniques to Deep Space Tasks, R.B. Ward, IEEE Transactions on Space Electronics and Telemetry, June 1964, P. 49-65.

A Comparison of Binary Delay - Lock Tracking Loop Implementations, Walter J. Gill, IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-2, No. 4, July 1966, P. 415-424.

Analysis of a Dithering Loop for PN code Tracking, Peter Hartmann, IEEE Transactions on Aerospace and Electronic Systems, January 1974, P. 2-9.

Double Dither Loop for Pseudonoise Code Tracking, Phillip M. Hopkins, IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-13, No. 6, November 1977, P. 644-650.

Spread Spectrum Texts/References:

Spread Spectrum Systems, Robert Dixon, 2nd edition, John Wiley and Sons, 1984. For those not familiar with spread spectrum systems read this book first.

Digital Communications and Spread Spectrum Systems, Rodger E. Ziemer and Roger L. Peterson, Macmillan Pub., 1984.

Modern Communications and Spread Spectrum, George R. Cooper and Clare D. McGillem, McGraw-Hill Inc.,1986.

Digital Communications, Satellite/Earth Station Engineering, Kamilo Feher, Prentice-Hall Inc., 1983.

Coherent Spread Spectrum Systems, J.K. Holmes, Wiley, 1982.

Other Books and Publications:

Microwave Transistor Amplifiers Analysis and Design, Guillermo Gonzalez, Prentice-Hall Inc., 1984.

RF Circuit Design, Chris Bowick, Howard W. Sams and Co., 1982.

Ferromagnetic Core Design and Application Handbook, Doug DeMaw, Prentice-Hall, 1981.

Communications Receivers Principles and Design, Ulrich L. Rohde and T.T.N. Bucher, McGraw-Hill Inc., 1988.

Phase Locked Loops, Roland E. Best, McGraw-Hill Inc., 1984.

IC Op-Amp Cookbook, 3rd edition, Walter G. Jung, Howard W. Sams and Co., 1986.

Active Filter Cookbook, Don Lancaster, Howard W. Sams and Co., 1975.

MECL System Design Handbook, 3rd Edition, William R. Blood Jr., Motorola Inc., 1980.

1982 Linear Databook, National Semiconductor Corp., Santa Clara, Calif.

A Handy "how to use" Guide for MAR Monolithic Drop-in Amplifiers, Mini Circuits Inc., Brooklyn, New York.

LIST OF FIGURES AND TABLES

Figures:

Tables:

SIMPLIFIED MODEL OF GPS L1 GENERATION

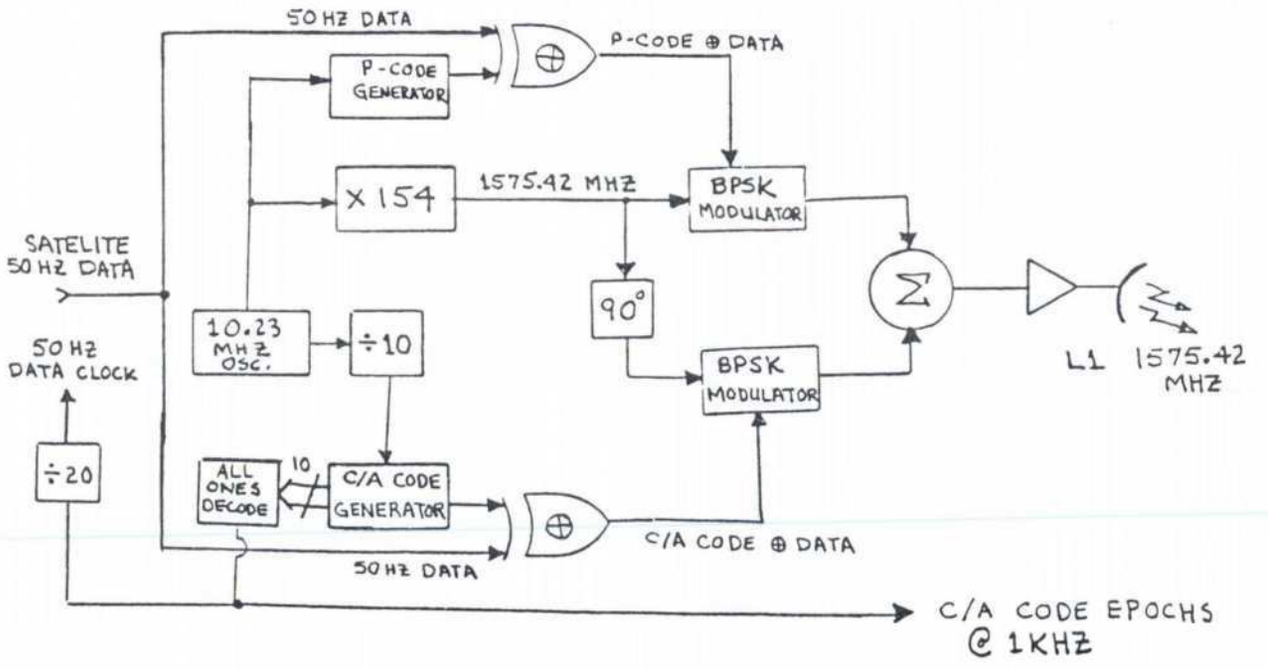


FIGURE 1

C/A CODE GENERATOR

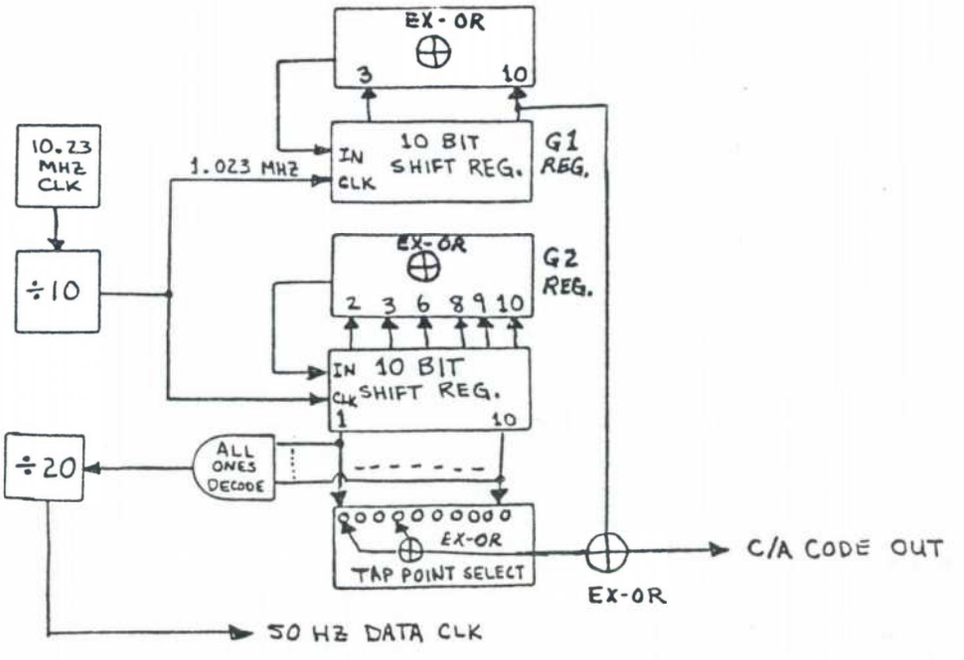


FIGURE 2

FIGURE 3
L1 POWER SPECTRUM

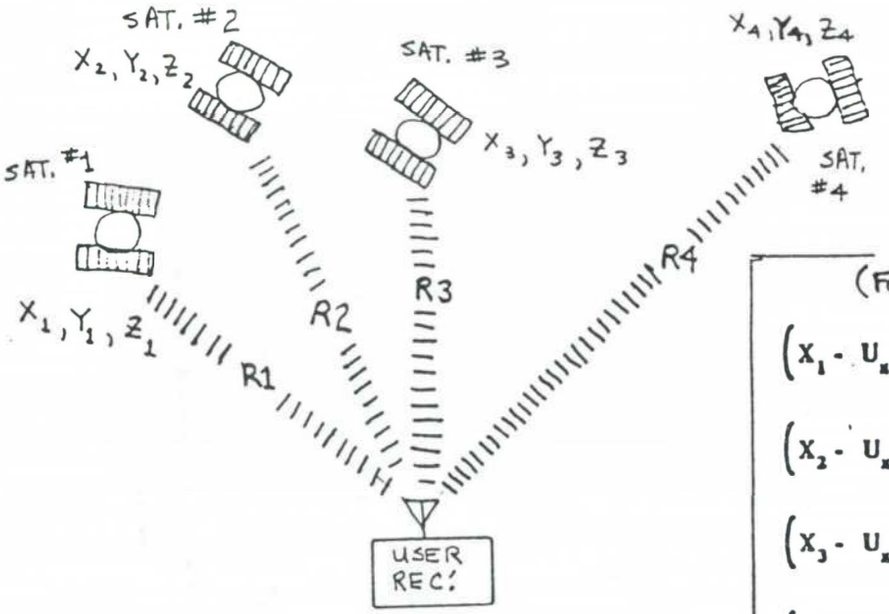
TABLE 1
C/A CODE SHIFT REG. TAP POINTS

SV ID No.	CPS PRN Signal No.	Code Phase Selection C/A ($G2_i$)	Code Phase Selection ($X2_i$)	Code Delay Chips C/A	Code Delay Chips P	First 10 Chips Octal C/A	First 12-Chips Octal P
1	1	2 ⊖ 6	1	5	1	1440	4444
* 2	2	3 ⊖ 7	2	6	2	1620	4000
** 3	3	4 ⊖ 8	3	7	3	1710	4222
4	4	5 ⊖ 9	4	8	4	1744	4333
5	5	1 ⊖ 9	5	17	5	1133	4377
* 6	6	2 ⊖ 10	6	18	6	1455	4355
7	7	1 ⊖ 8	7	139	7	1131	4344
8	8	2 ⊖ 9	8	140	8	1454	4340
* 9	9	3 ⊖ 10	9	141	9	1626	4342
10	10	2 ⊖ 3	10	251	10	1504	4343
* 11	11	3 ⊖ 4	11	252	11	1642	
** 12	12	5 ⊖ 6	12	254	12	1750	
** 13	13	6 ⊖ 7	13	255	13	1764	
** 14	14	7 ⊖ 8	14	256	14	1772	
15	15	8 ⊖ 9	15	257	15	1775	
* 16	16	9 ⊖ 10	16	258	16	1776	
17	17	1 ⊖ 4	17	469	17	1156	
18	18	2 ⊖ 5	18	470	18	1467	
19	19	3 ⊖ 6	19	471	19	1633	
20	20	4 ⊖ 7	20	472	20	1715	
21	21	5 ⊖ 8	21	473	21	1746	
22	22	6 ⊖ 9	22	474	22	1763	
23	23	1 ⊖ 3	23	509	23	1063	
24	24	4 ⊖ 6	24	512	24	1706	
25	25	5 ⊖ 7	25	513	25	1743	
26	26	6 ⊖ 8	26	514	26	1761	
27	27	7 ⊖ 9	27	515	27	1770	
28	28	8 ⊖ 10	28	516	28	1774	
29	29	1 ⊖ 6	29	859	29	1127	
30	30	2 ⊖ 7	30	860	30	1453	
31	31	3 ⊖ 8	31	861	31	1625	
32	32	4 ⊖ 9	32	862	32	1712	
: :	33	5 ⊖ 10	33	863	33	1745	
: :	34*	4 ⊖ 10	34	950	34	1713	
: :	35	1 ⊖ 7	35	947	35	1134	
: :	36	2 ⊖ 8	36	948	36	1458	
: :	37*	4 ⊖ 10	37	950	37	1713	4343

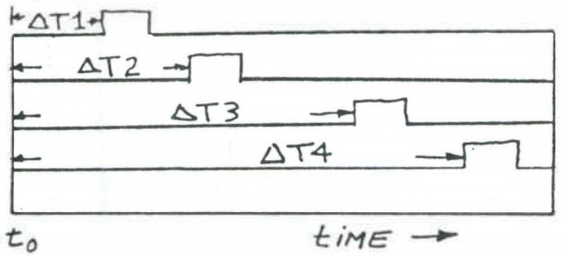
*C/A codes 34 and 37 are common.
**PRN sequences 33 through 37 are reserved for other uses (e.g. ground transmitters).

FROM ICD-GPS-200
* ACTIVE SATELLITES AT THE TIME OF THIS PAPER

FIGURE 4. POSITION MEASUREMENT

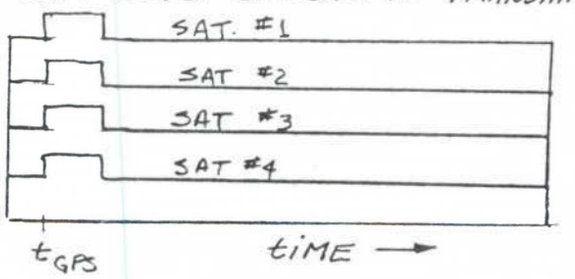


MEASURE FOUR DELAYS & COMPUTE PSEUDO RANGES



C/A EPOCHS RECEIVED BY USER ARE DELAYED BY PATH LENGTHS R1 - R4.

C/A CODE EPOCHS AT TRANSMITTERS



COMPUTE POSITION COORDINATES
(FOUR EQUATIONS with FOUR UNKNOWNS

$$(X_1 - U_x)^2 + (Y_1 - U_y)^2 + (Z_1 - U_z)^2 = (R_1 - C_B)^2$$

$$(X_2 - U_x)^2 + (Y_2 - U_y)^2 + (Z_2 - U_z)^2 = (R_2 - C_B)^2$$

$$(X_3 - U_x)^2 + (Y_3 - U_y)^2 + (Z_3 - U_z)^2 = (R_3 - C_B)^2$$

$$(X_4 - U_x)^2 + (Y_4 - U_y)^2 + (Z_4 - U_z)^2 = (R_4 - C_B)^2$$

SOLVE FOR USER'S POSITION COORDINATES
(U_x, U_y, U_z) AND CLOCK BIAS, C_B

C_B IS CLOCK BIAS RANGING ERROR
$= C\Delta T_B$

ΔT_B = USER CLOCK TIMING ERROR

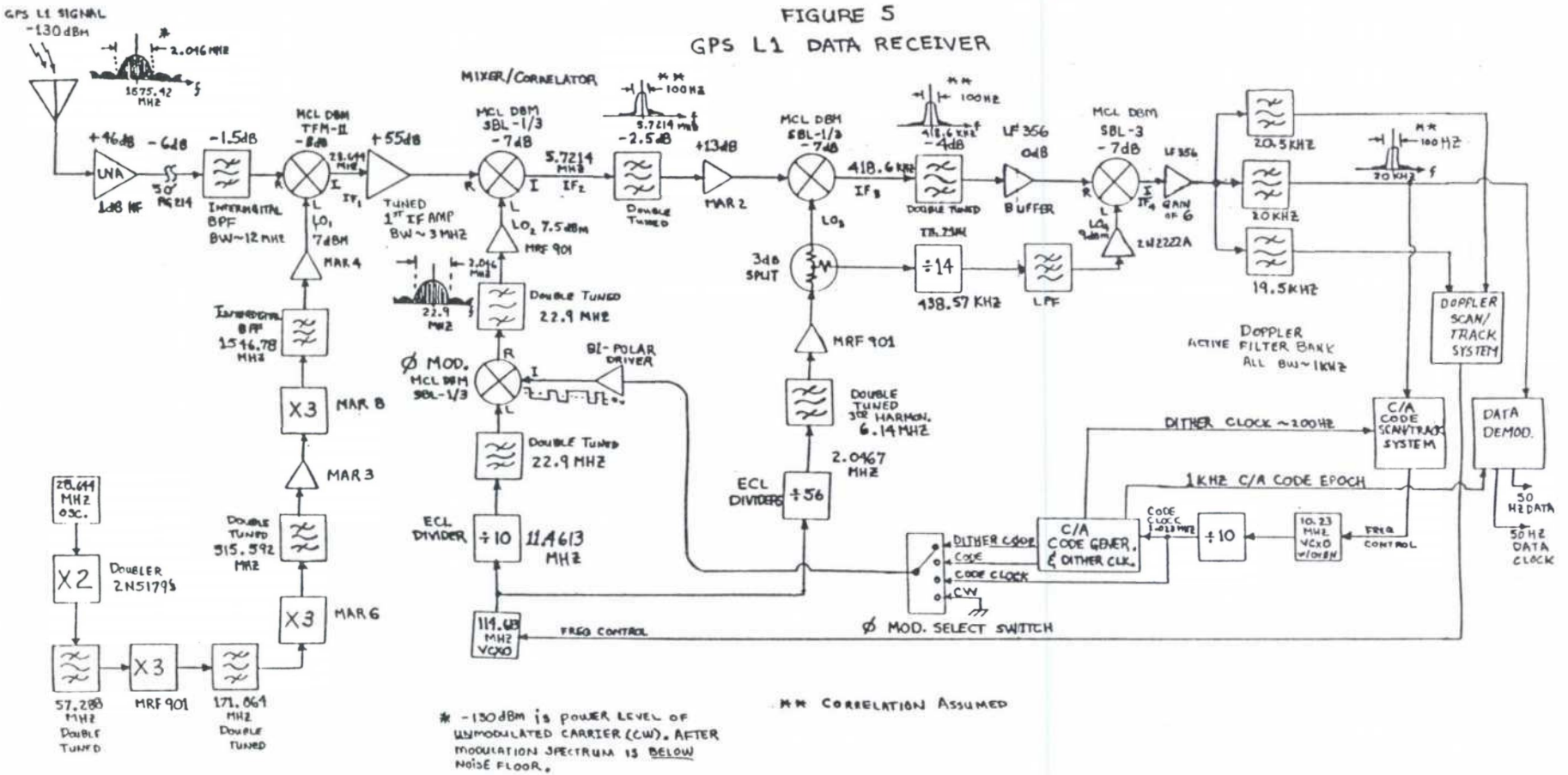
PSEUDO RANGES

$R1 = C \times \Delta T1$
$R2 = C \times \Delta T2$
$R3 = C \times \Delta T3$
$R4 = C \times \Delta T4$

C is SPEED OF LIGHT

FIGURE 5

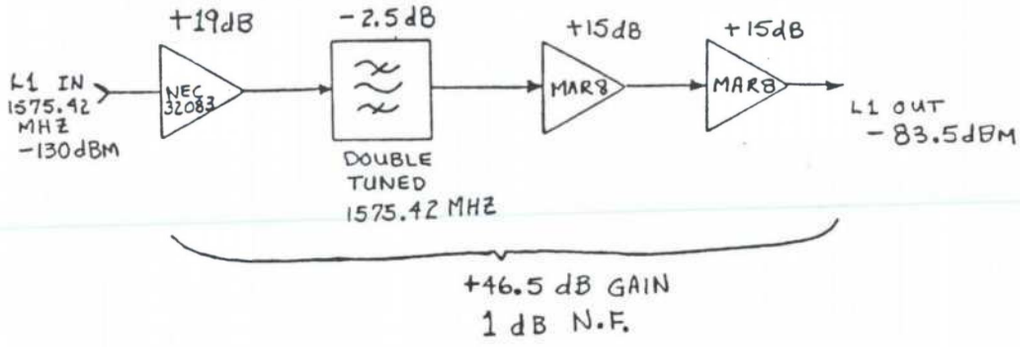
GPS L1 DATA RECEIVER

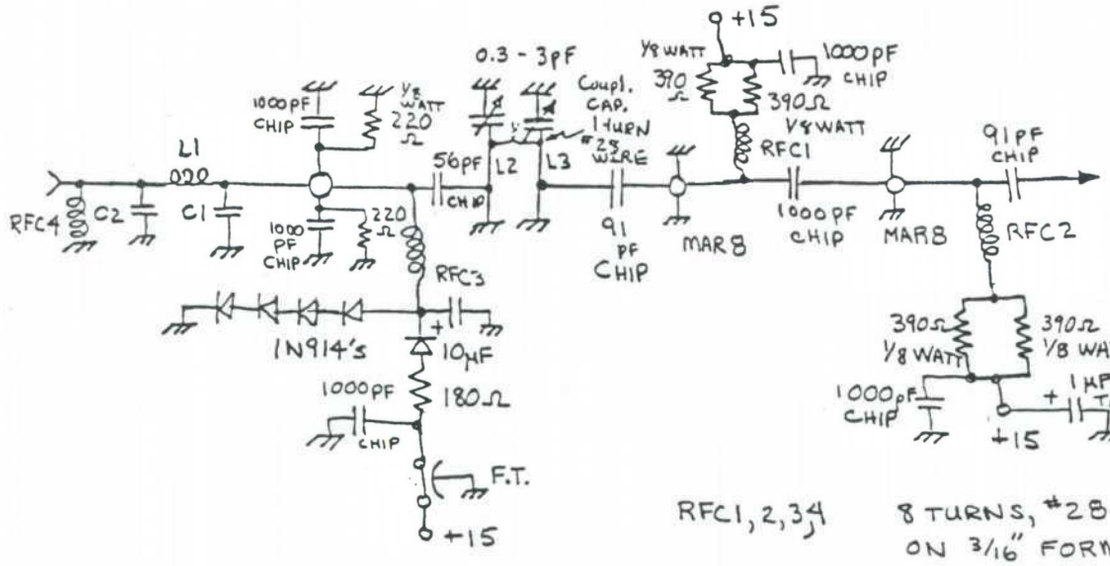


* -130dBm is power level of unmodulated carrier (CW). After modulation spectrum is BELOW noise floor.

** Correlation Assumed

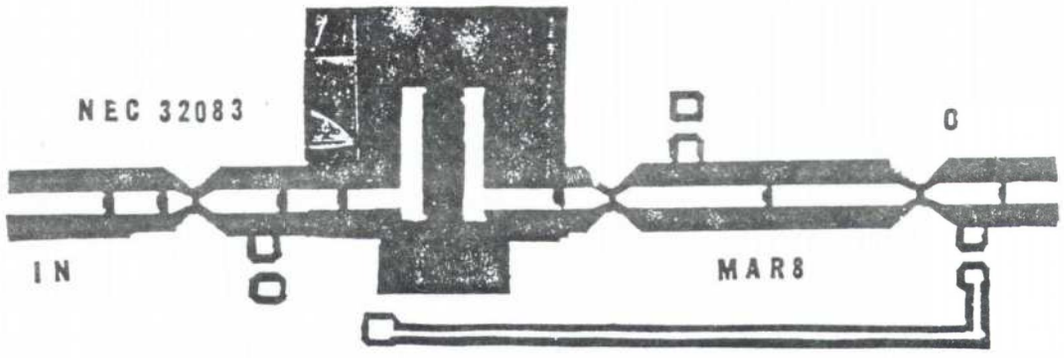
FIGURE 6
PREAMP BLOCK DIAGRAM & SCHEMATIC

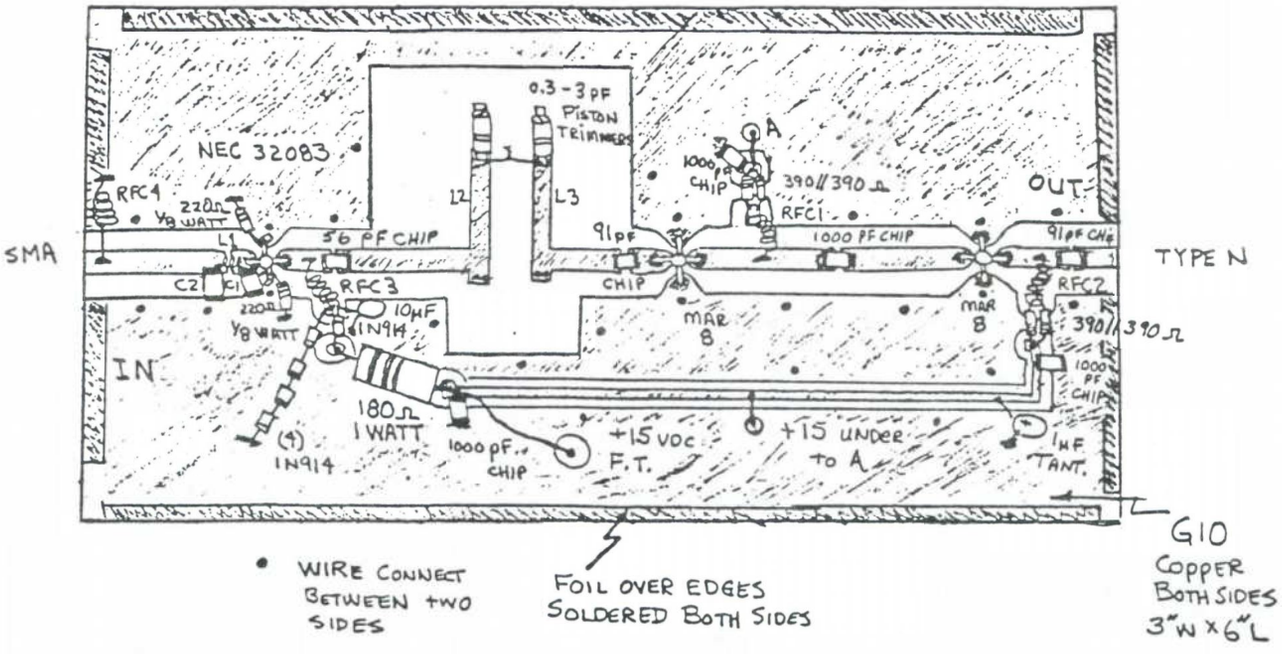


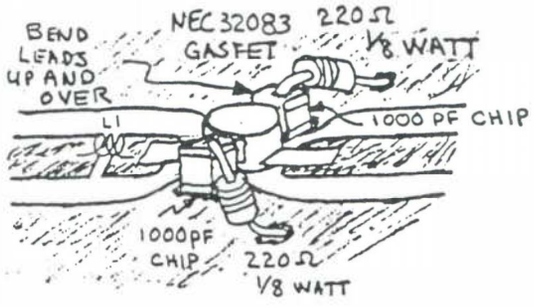


RFC1, 2, 3, 4	8 TURNS, #28 ENAM. WIRE ON 3/16" FORM
C1	2.2 pF, CHIP
C2	0.3 pF, CHIP
L1	3 TURNS, #28 ENAM. WIRE ON #50 DRILL (0.07" DIA)

FIGURE 7
LAYOUT & ARTWORK FOR PREAMP

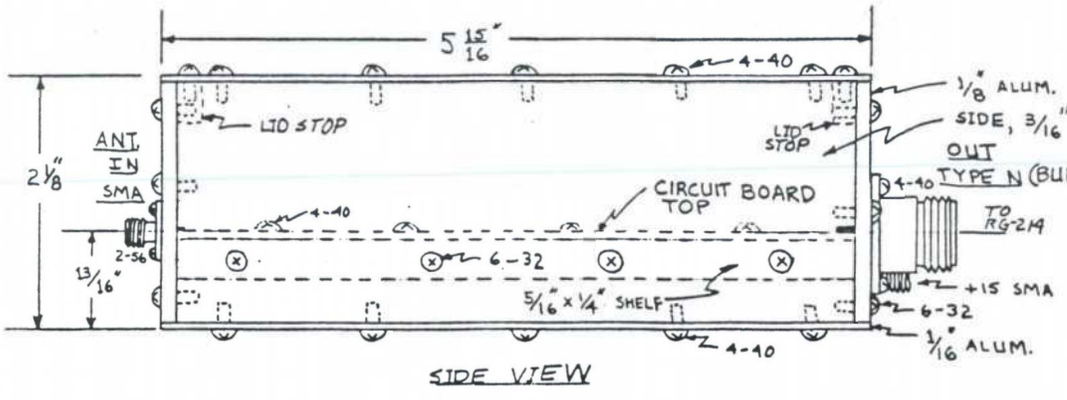






DETAIL OF NEC 32083

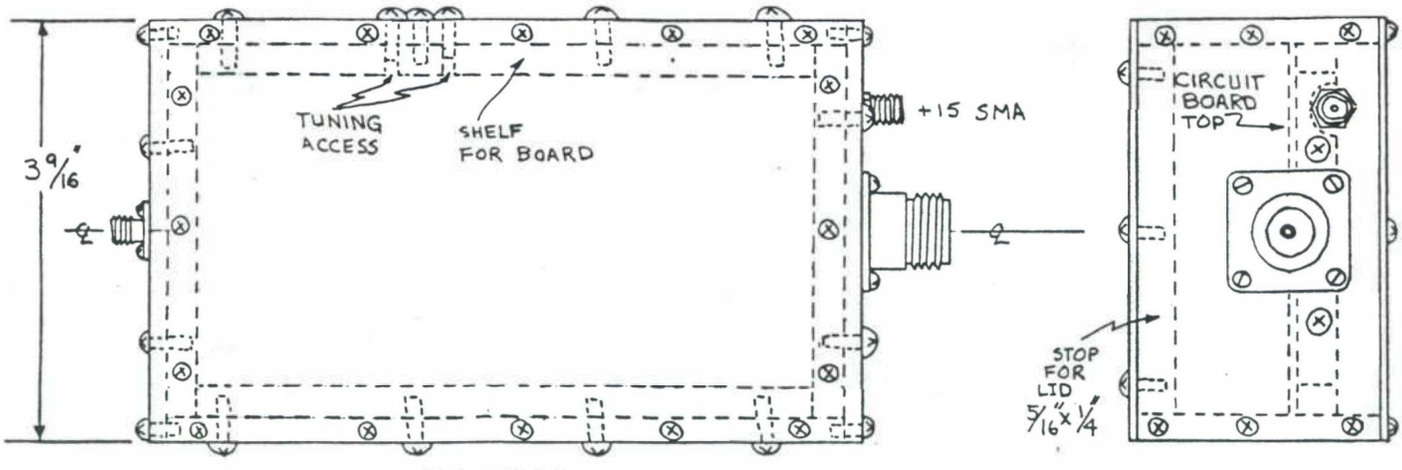
MOUNTING TO CHIP CAPS; C1 & C2 ARE NOT SHOWN



SIDE VIEW

FIGURE 8

PREAMP BOX

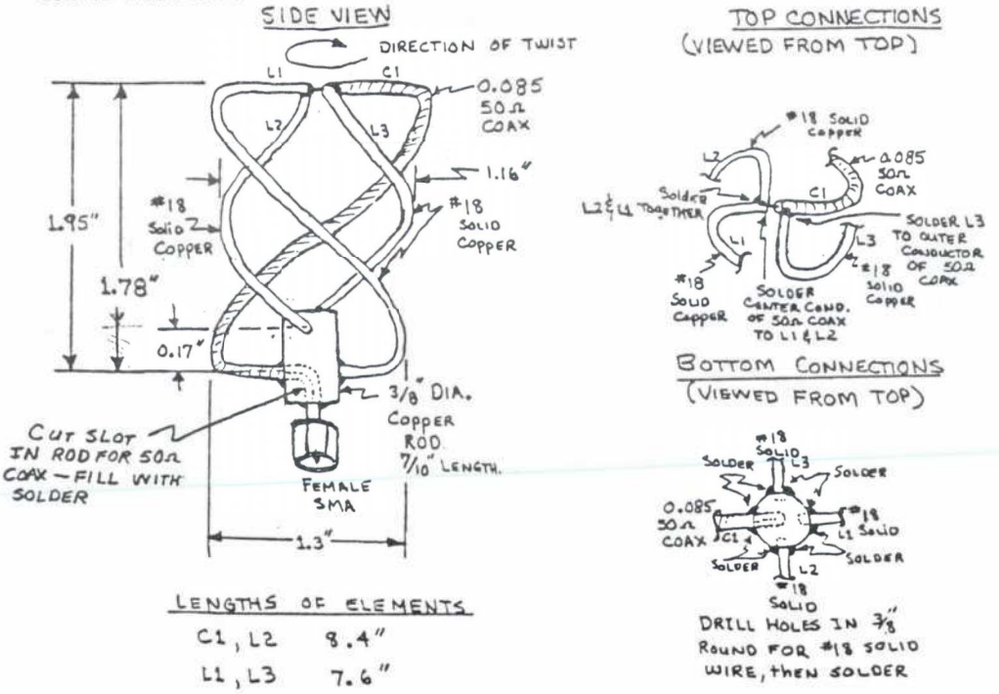


TOP VIEW

END VIEW

FIGURE 9

QUADRIFILAR ANTENNA - FROM ARRL ANTENNA BOOK, 15TH ED.

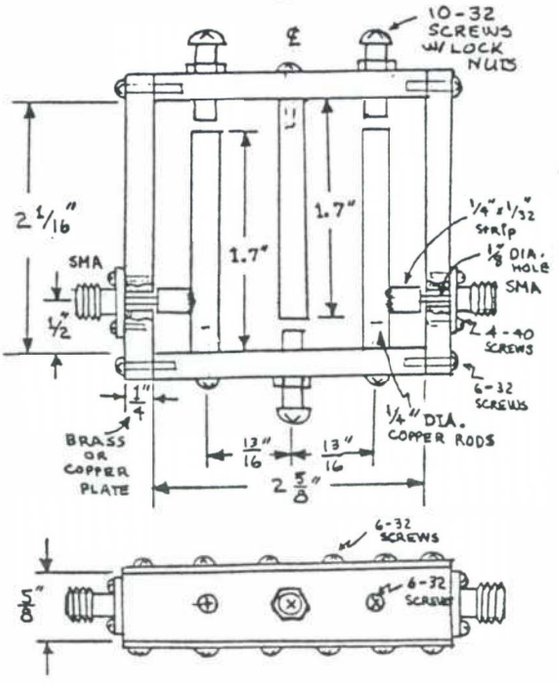


LENGTHS OF ELEMENTS

C1, L2	8.4"
L1, L3	7.6"

FIGURE 10

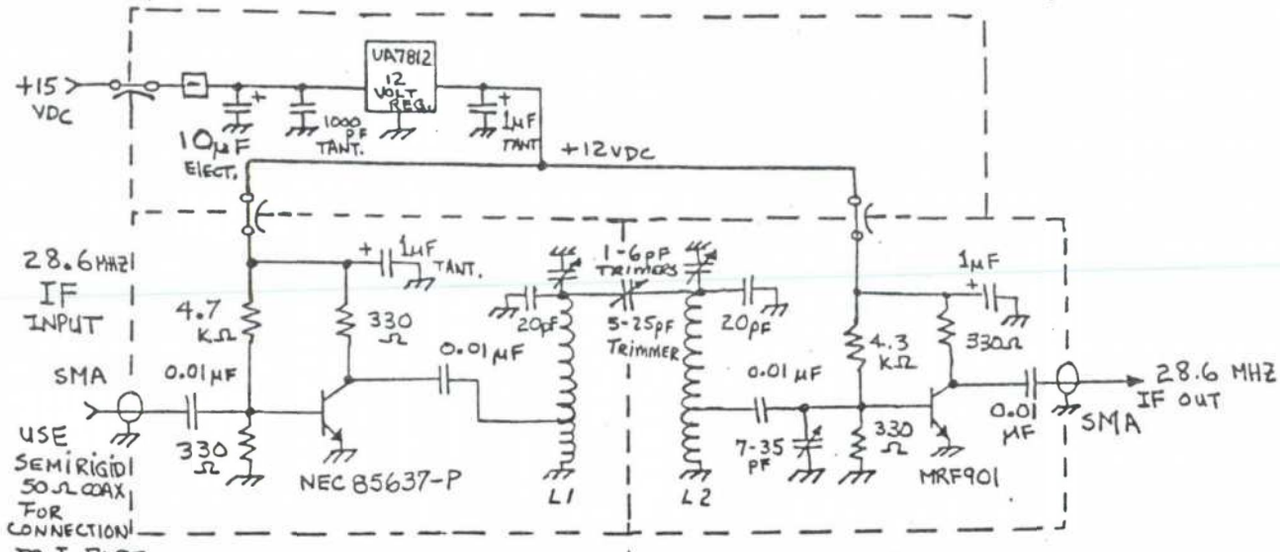
1575.42 BPF, INTERDIGITAL DETAIL*



* SEE ARRL HANDBOOK 1988
OR VHF/UHF HANDBOOK, RSGB

FIGURE 11

28.6 MHZ 1ST IF TUNED AMP. (55dB GAIN, 3MHZ 3dB B.W.)



L1, L2 17 TURNS, #18 WIRE, 3/8" I.D.
TAPPED 5 TURNS FROM BOTTOM

1ST L.O.

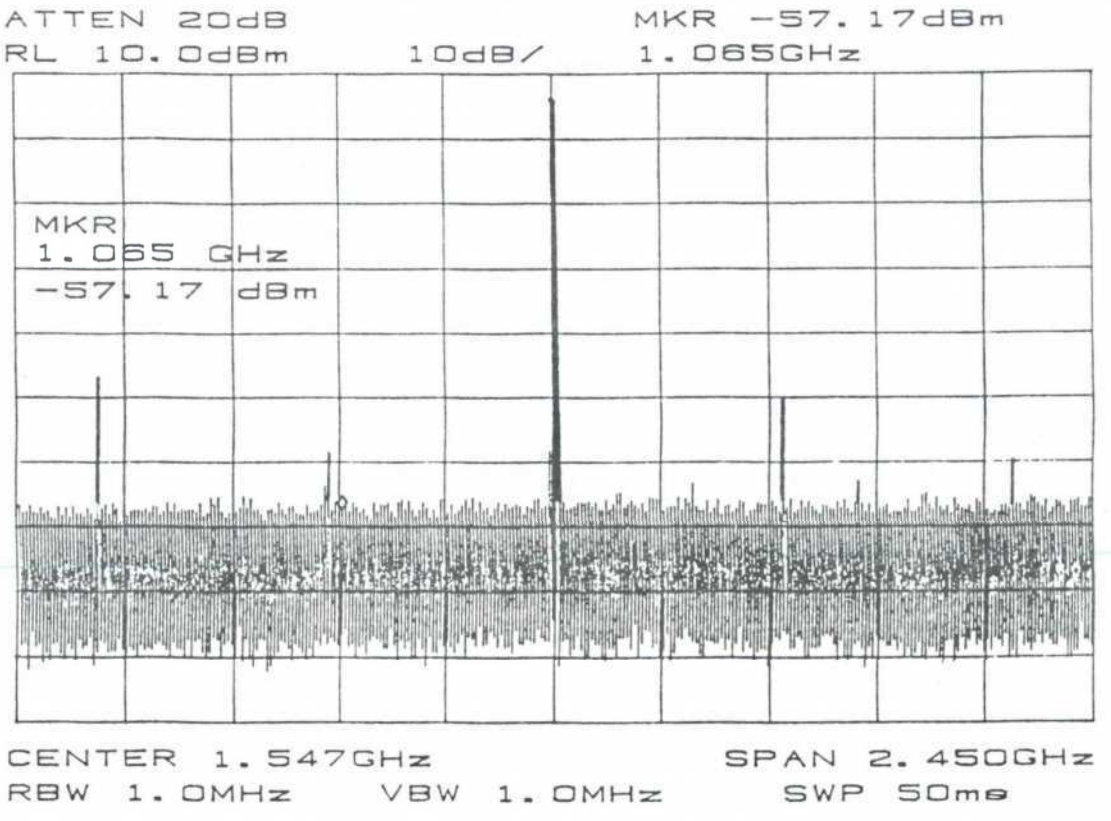


FIGURE 12
1ST L.O. OUTPUT

1ST L.O.

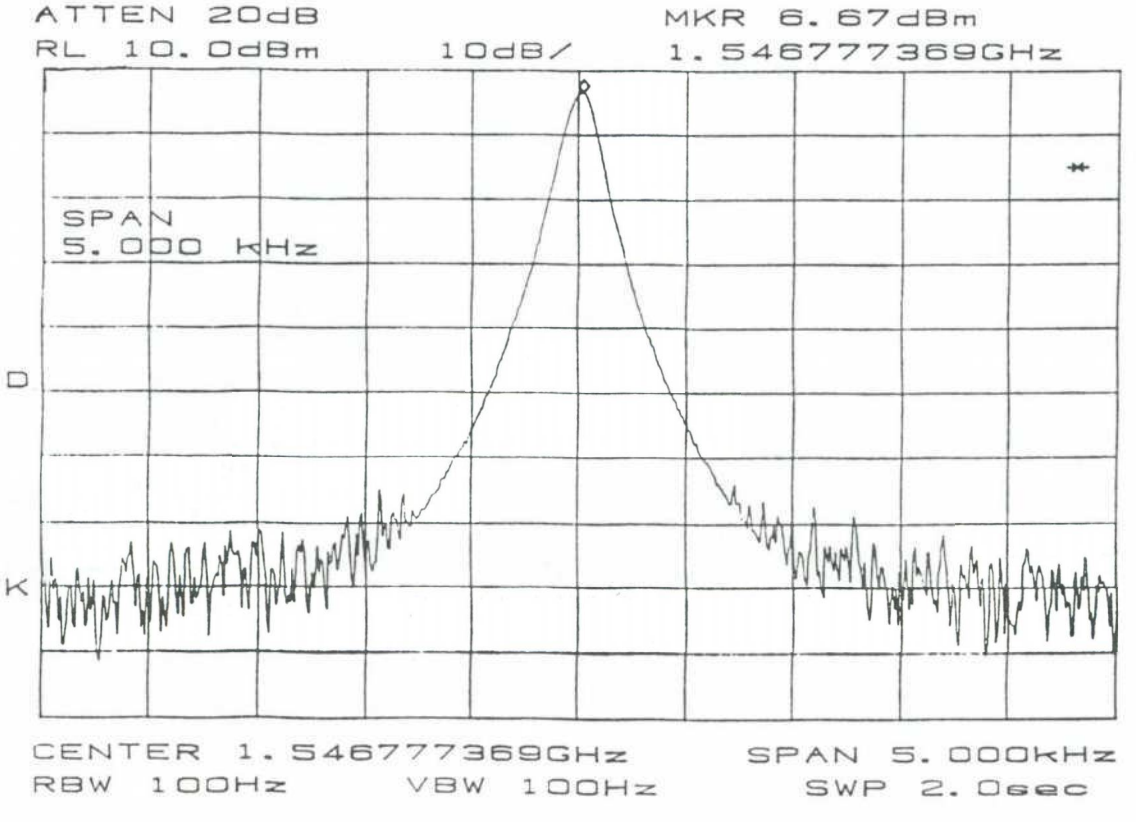
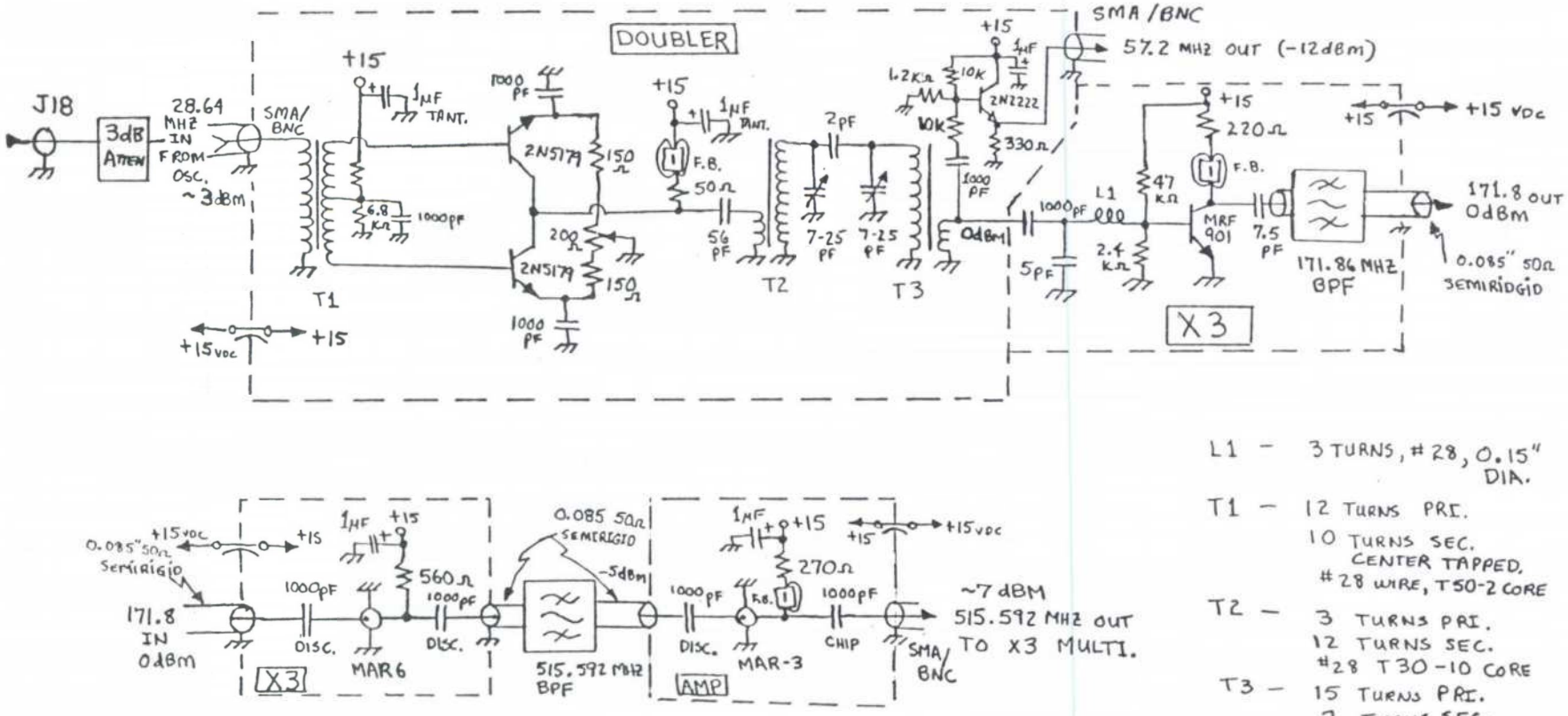


FIGURE 13 1ST L.O. MULTI. (X2, X3 & X3) TO 515 MHZ



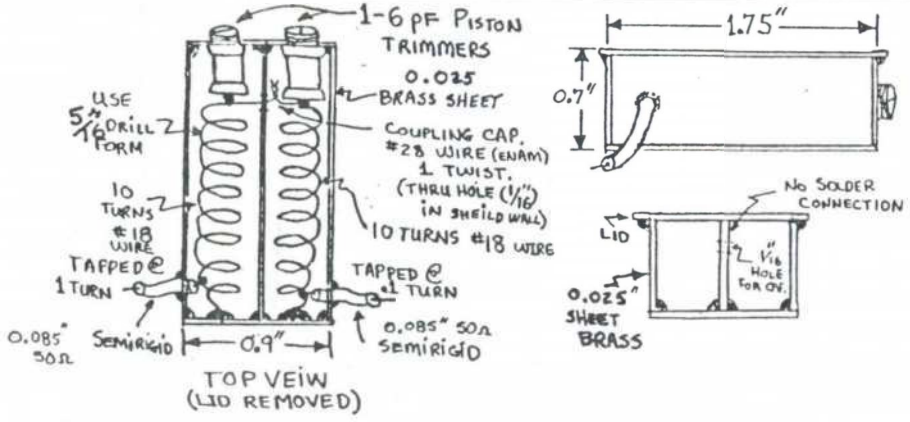
L1 – 3 TURNS, #28, 0.15" DIA.

T1 – 12 TURNS PRI.
10 TURNS SEC. CENTER TAPPED.
#28 WIRE, T50-2 CORE

T2 – 3 TURNS PRI.
12 TURNS SEC.
#28 T30-10 CORE

T3 – 15 TURNS PRI.
2 TURNS SEC.
#28 T30-10 CORE.

171 BPF DETAIL



515 BPF DETAIL

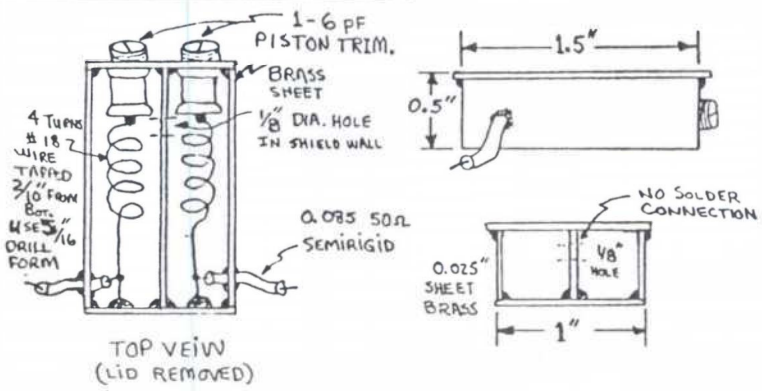


FIGURE 15. 1ST L.O. 515 TO 1546 ARTWORK SHOWN IN NEGATIVE.

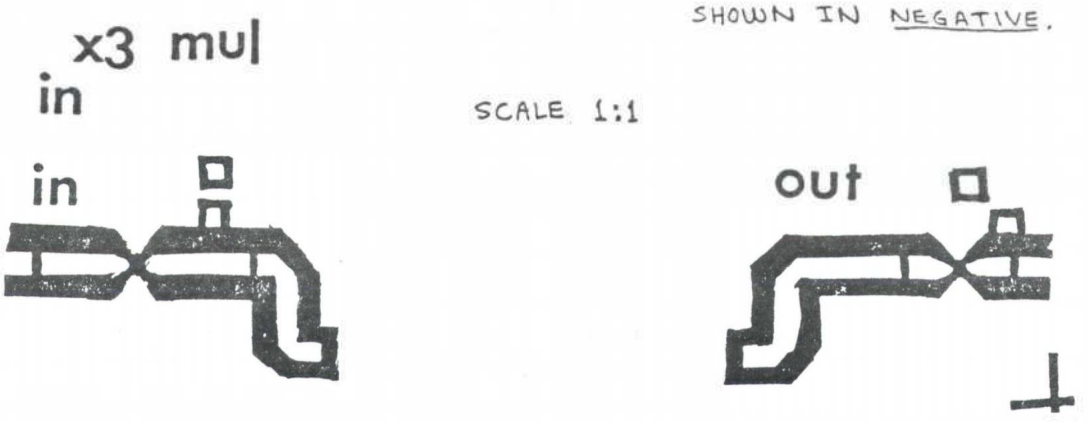
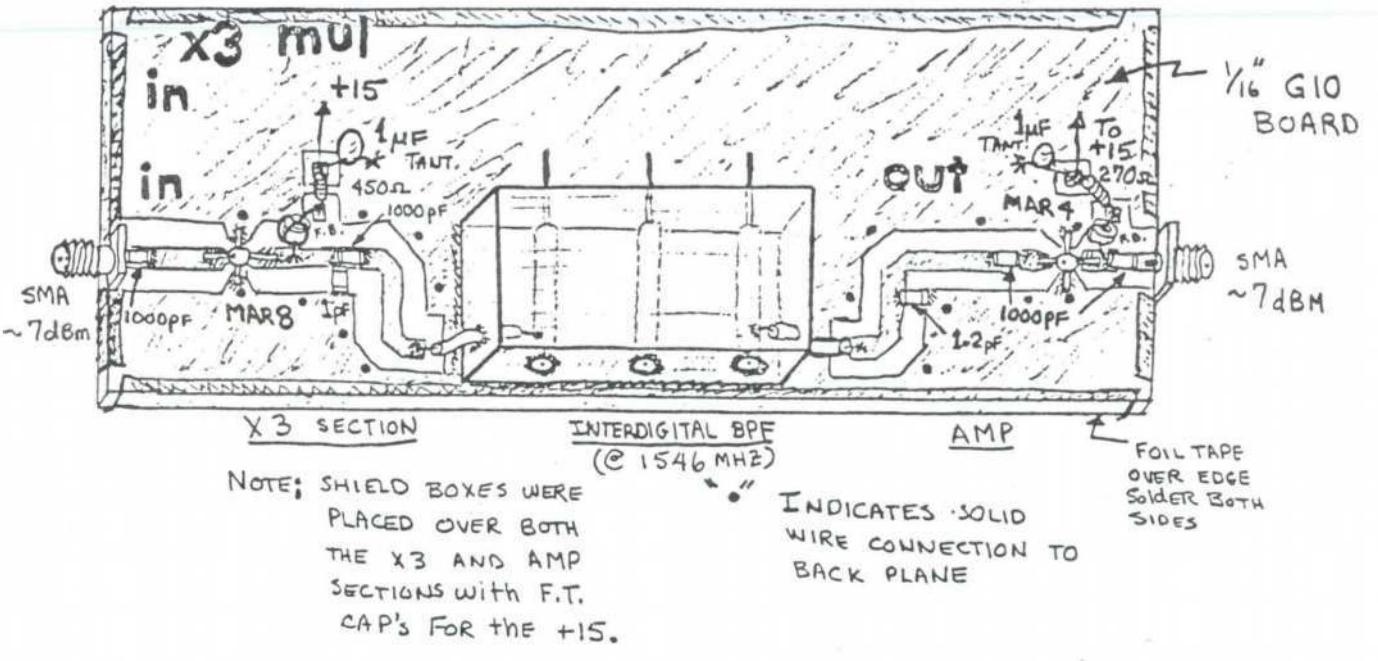


FIGURE 16. LAYOUT OF 1ST L.O. 515 TO 1546 x3 MULT.



1546 MHZ BPF DETAIL (SEE ARRL 1988 HANDBOOK)

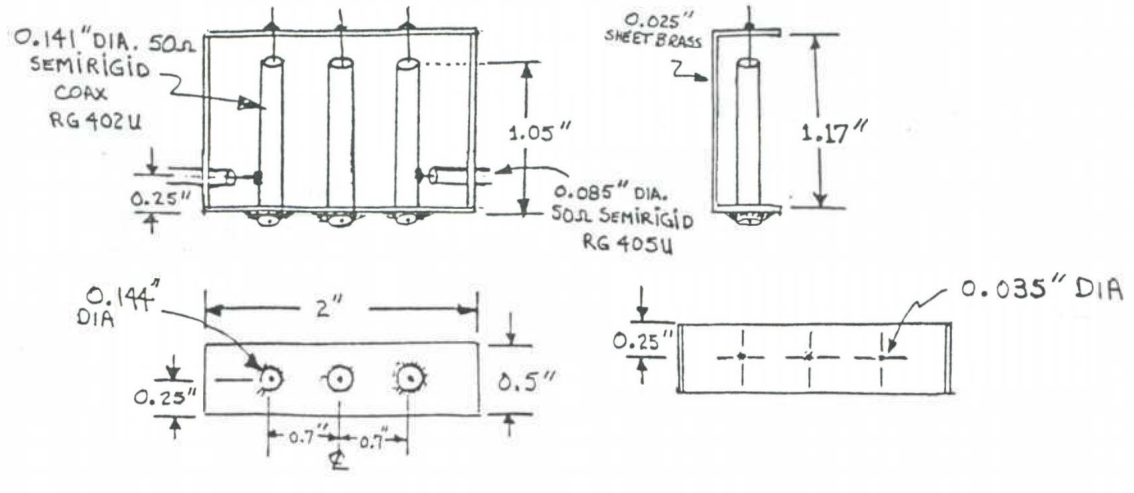
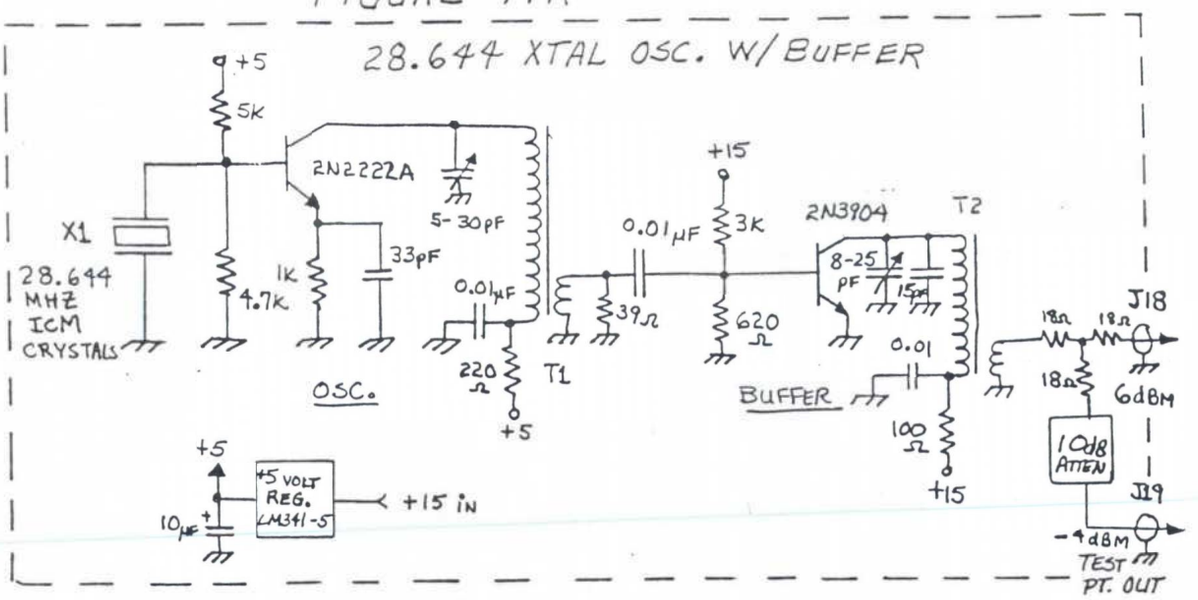


FIGURE 17A



T1 15/1 TURNS T-37-2 CORE

T2 15/2 TURNS T-37-2 CORE

FIGURE 17B

XTAL DOWNCONVERTER

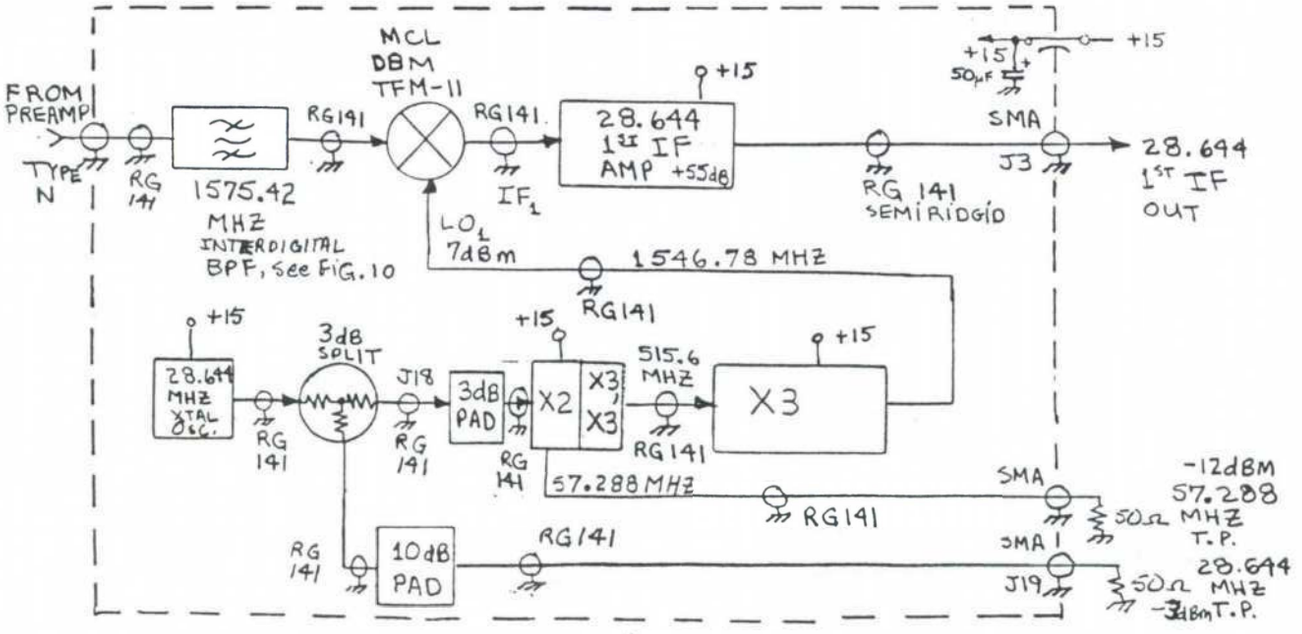


FIGURE 18

2ND, 3RD, 4TH LO's & IF's CIRCUIT(S)

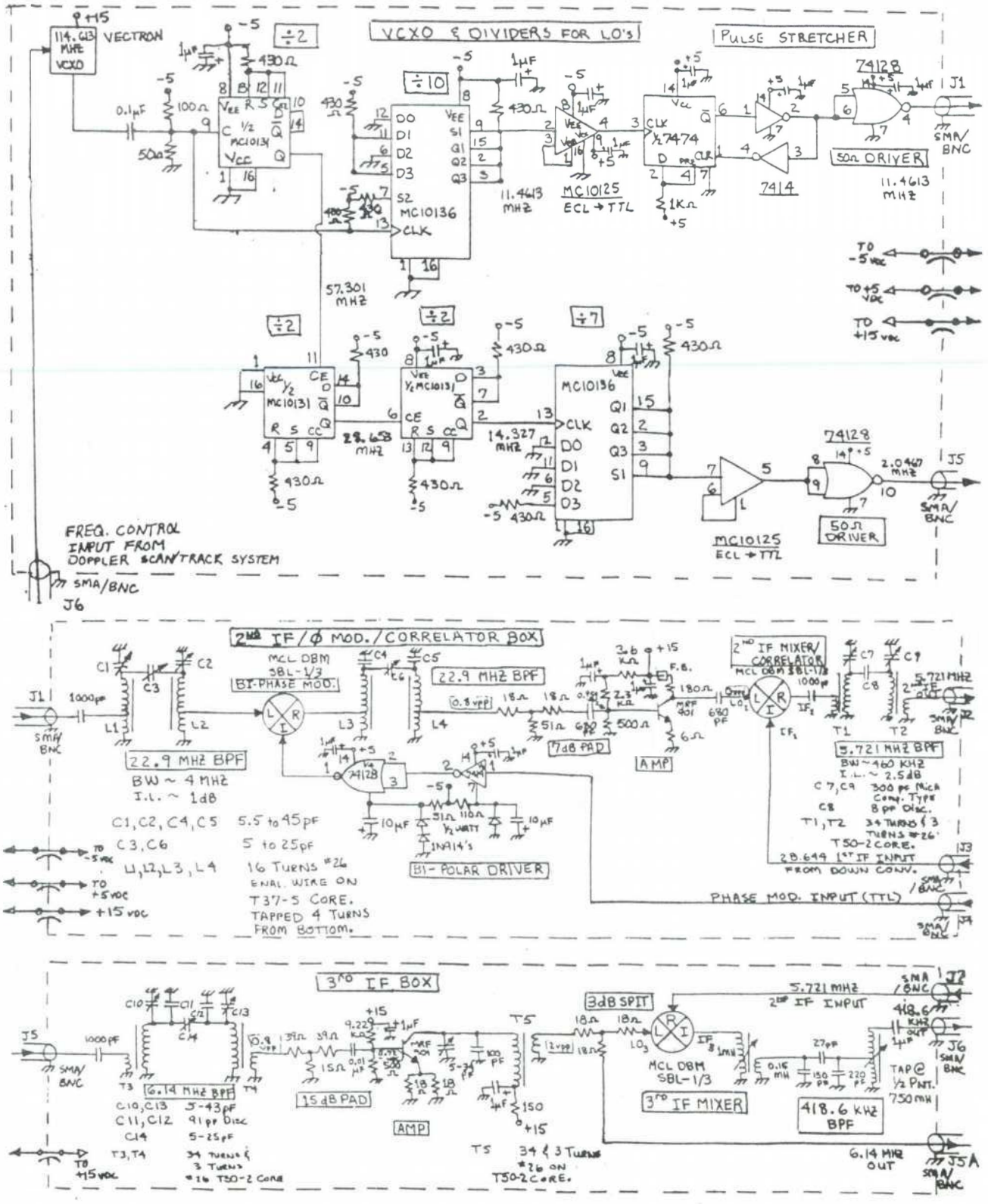


FIGURE 18 CONT.

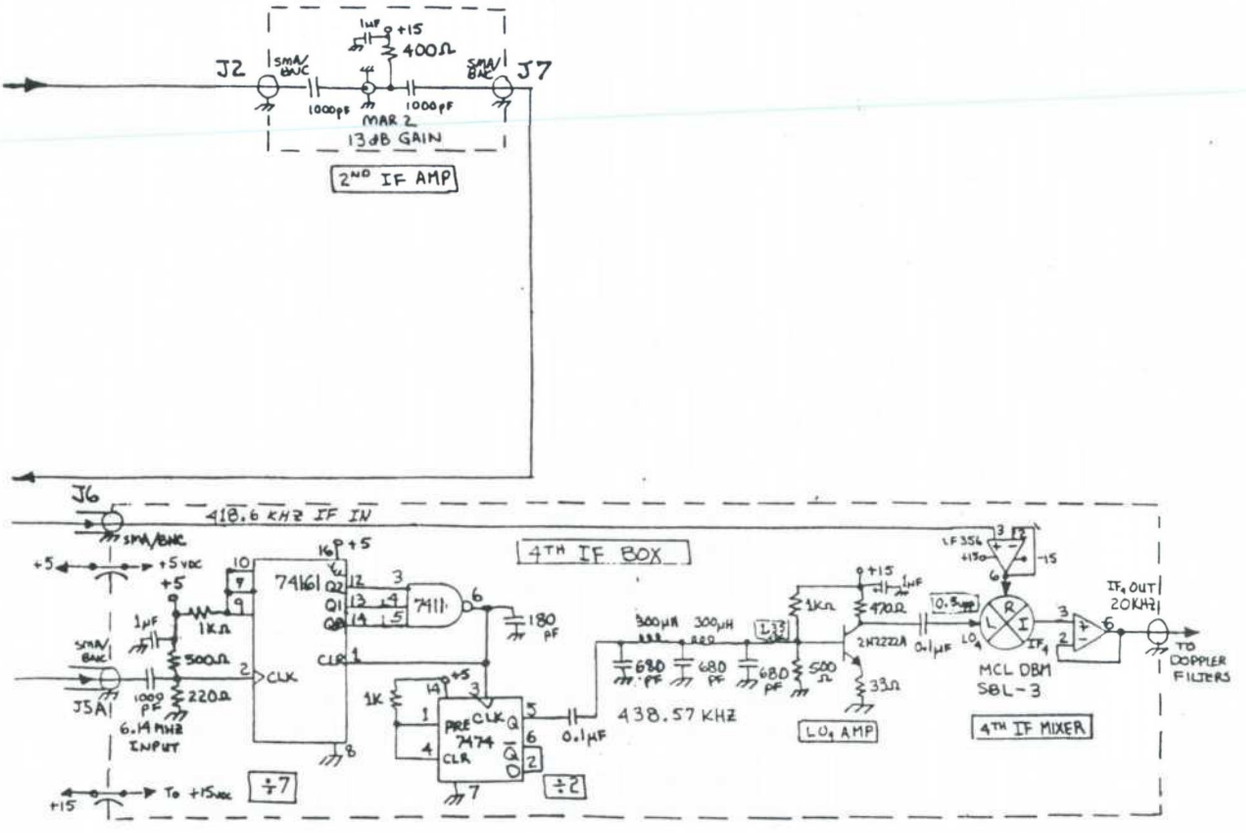
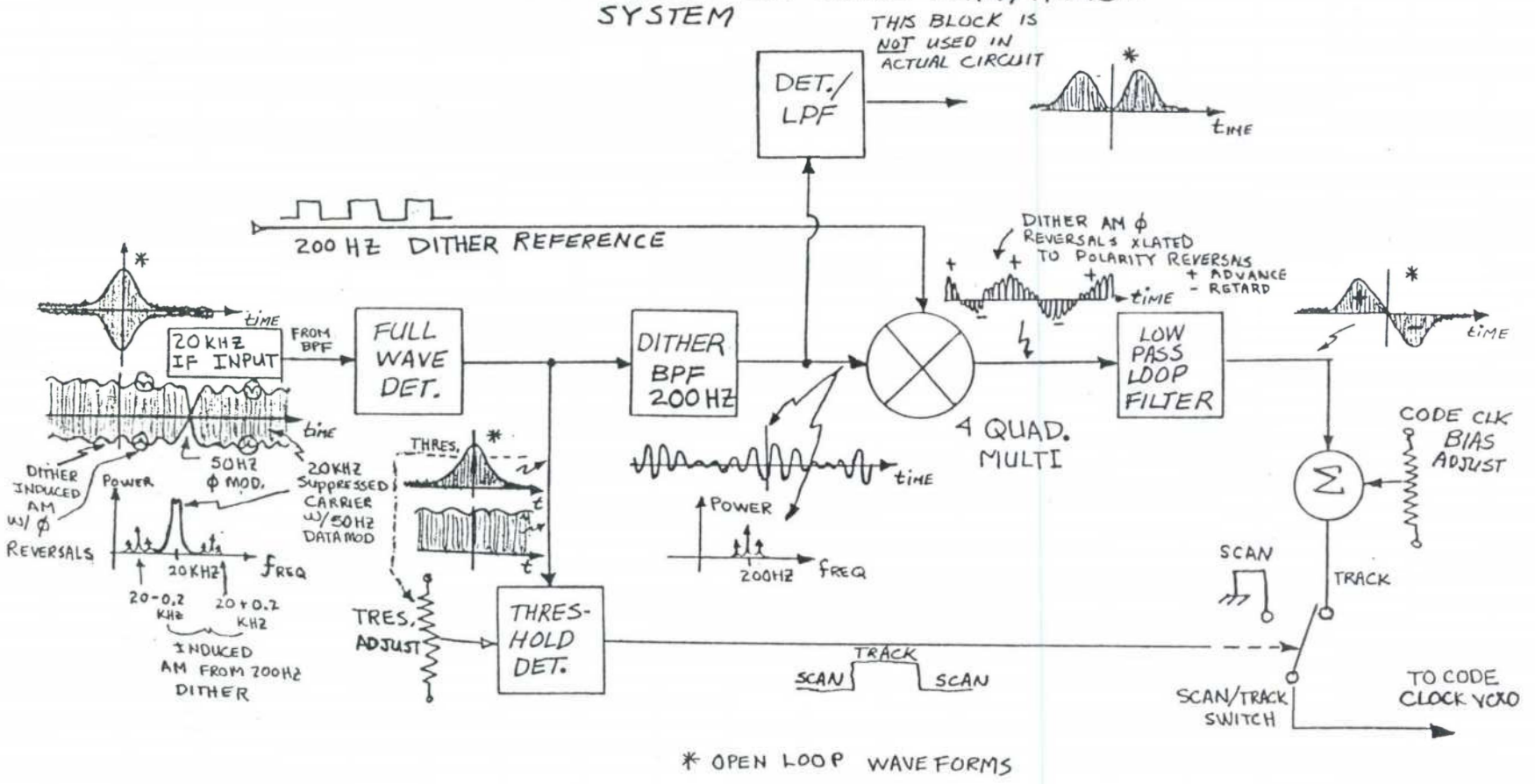


FIGURE 19

BLOCK DIAGRAM C/A CODE SCAN/TRACK SYSTEM



* OPEN LOOP WAVEFORMS

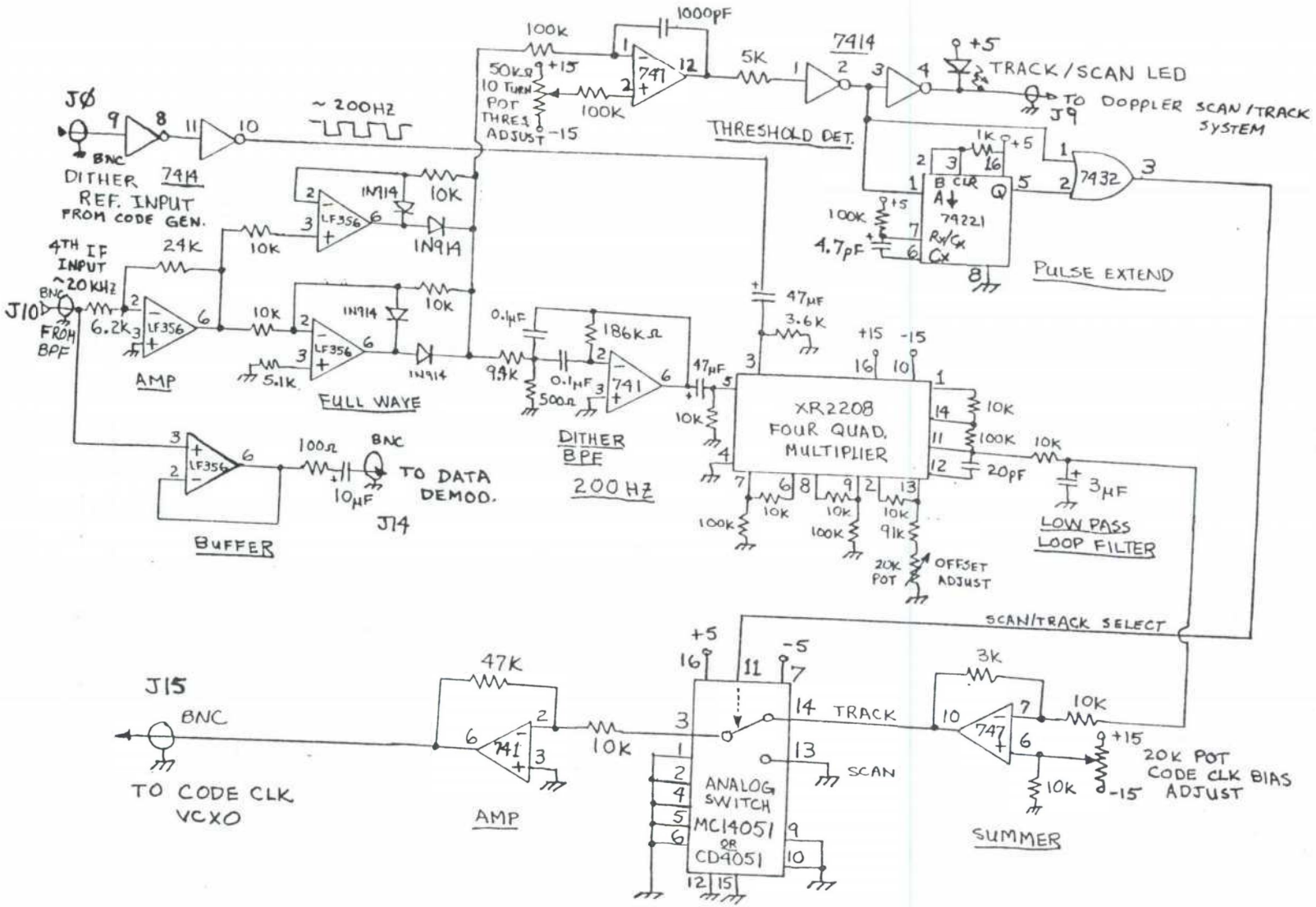


FIGURE 20.
C/A CODE SCAN/TRACK CIRCUIT

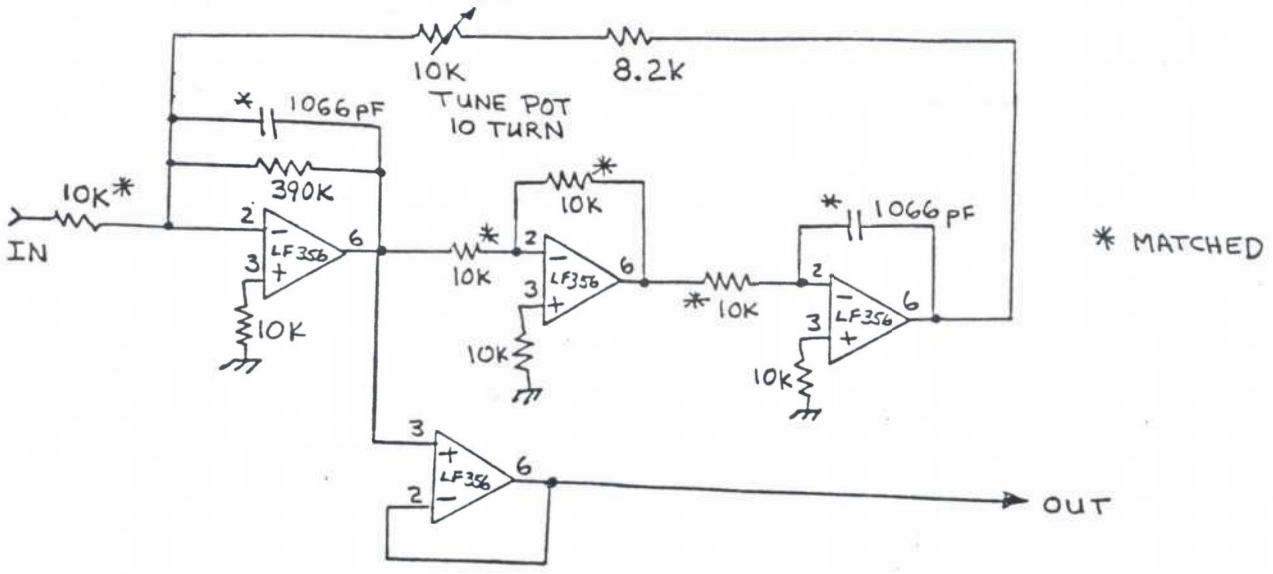


FIGURE 21

ACTIVE 20 KHZ BPF - SEE OP AMP COOKBOOK

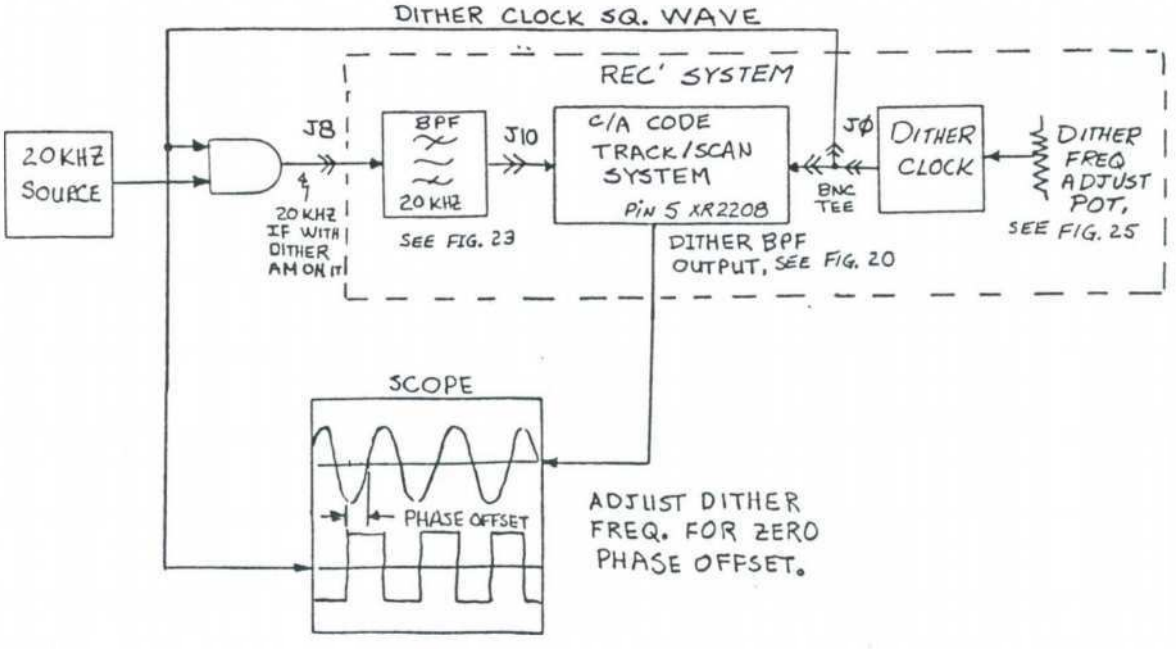


FIGURE 22

DITHER CLOCK TUNING SETUP

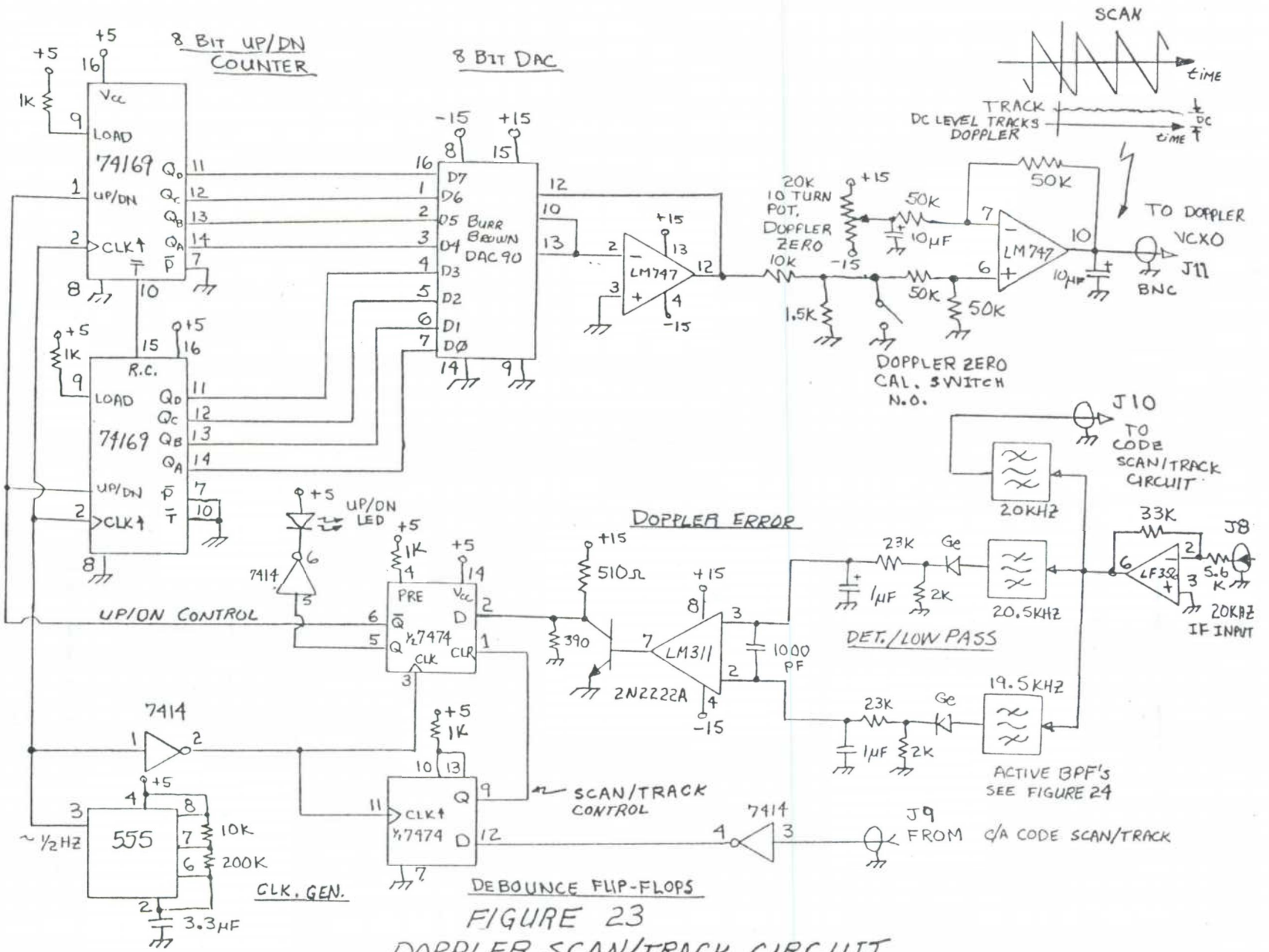


FIGURE 23
DOPPLER SCAN/TRACK CIRCUIT

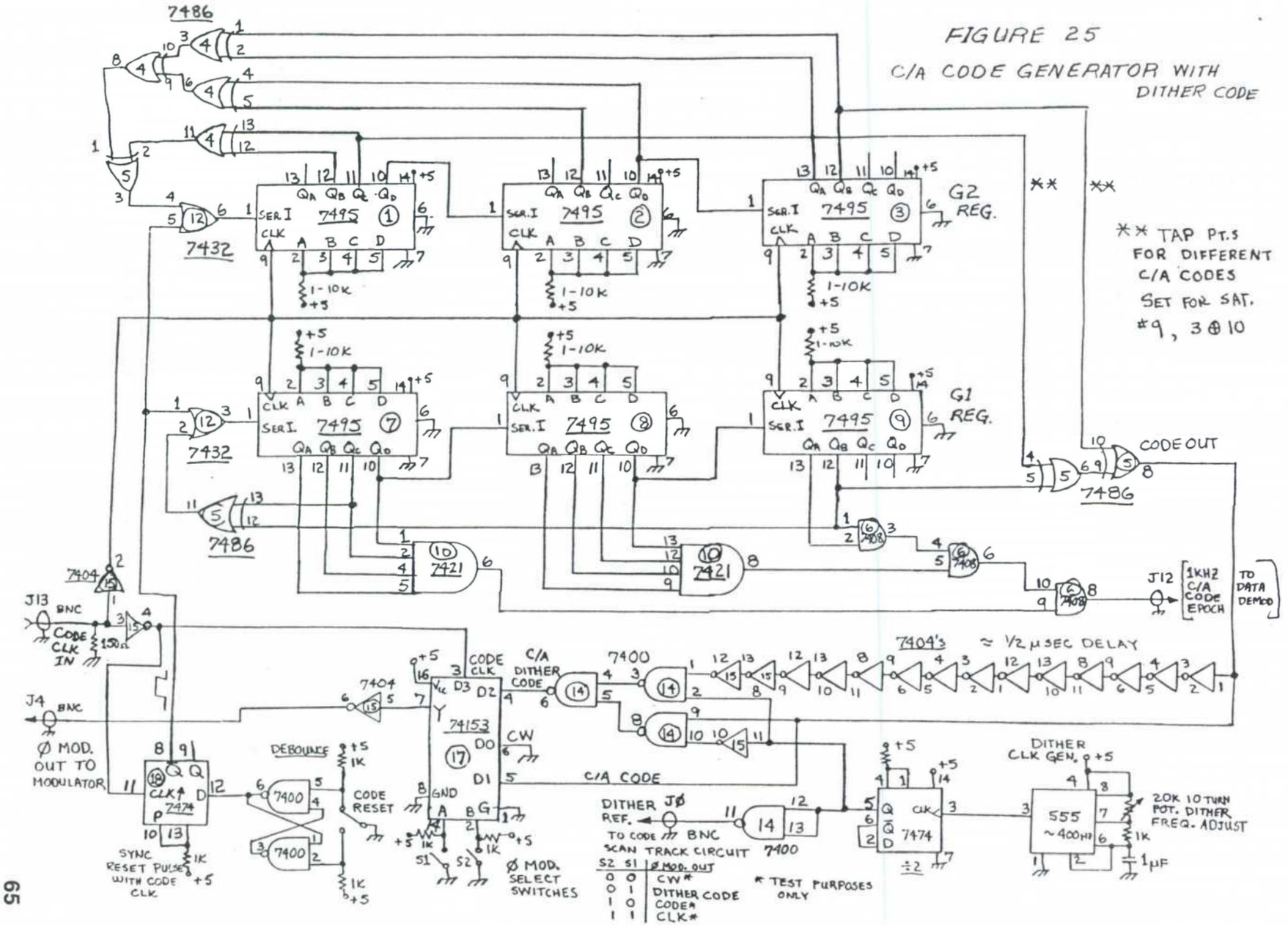


FIGURE 25

C/A CODE GENERATOR WITH DITHER CODE

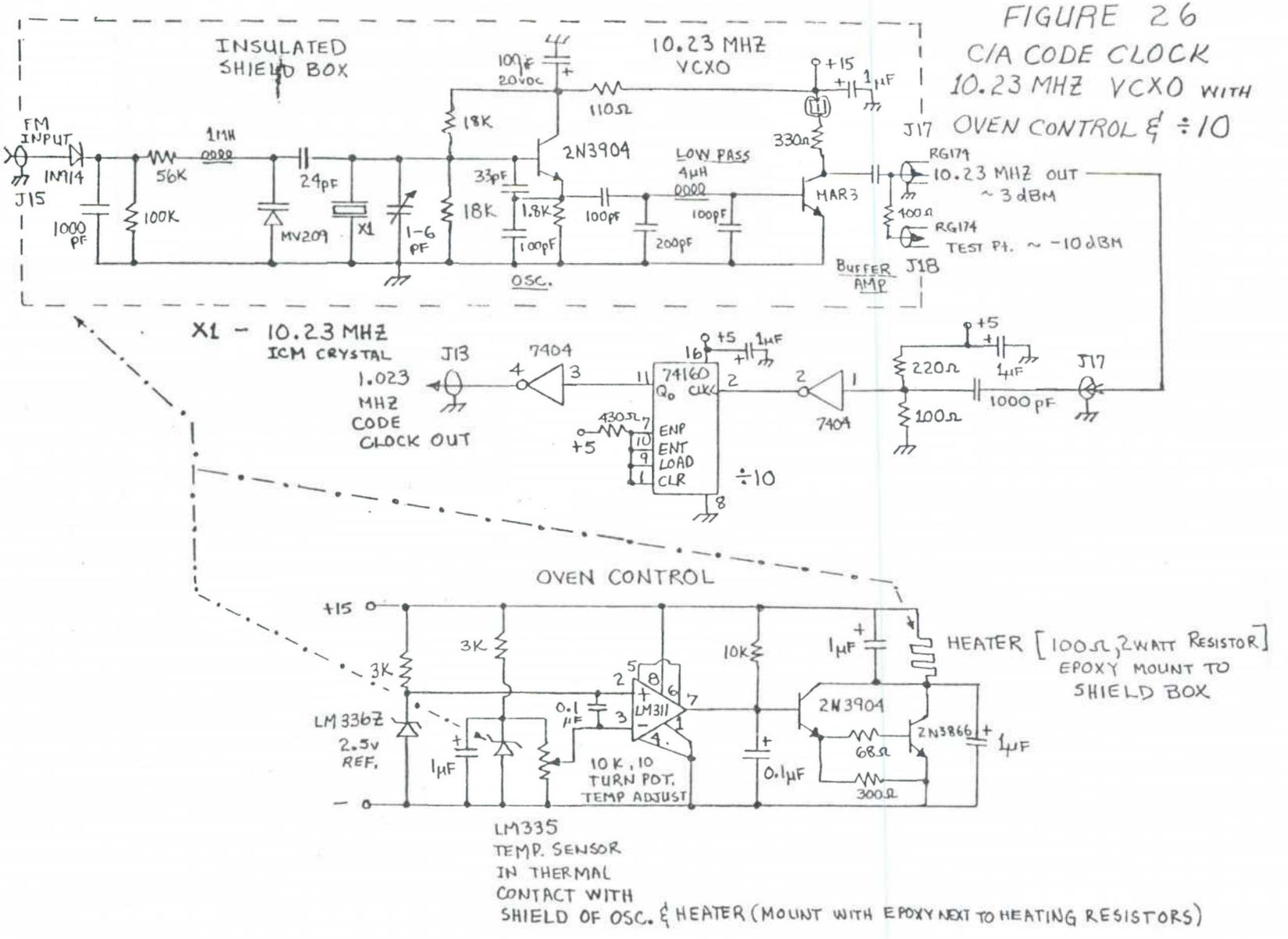


FIGURE 26
C/A CODE CLOCK
10.23 MHZ VCXO WITH
OVEN CONTROL & ÷10

FIGURE 27

DATA DEMODULATOR BLOCK DIAGRAM

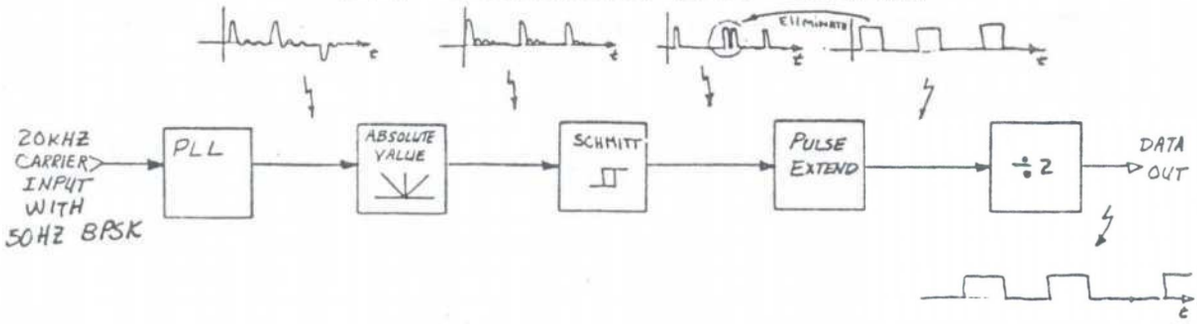
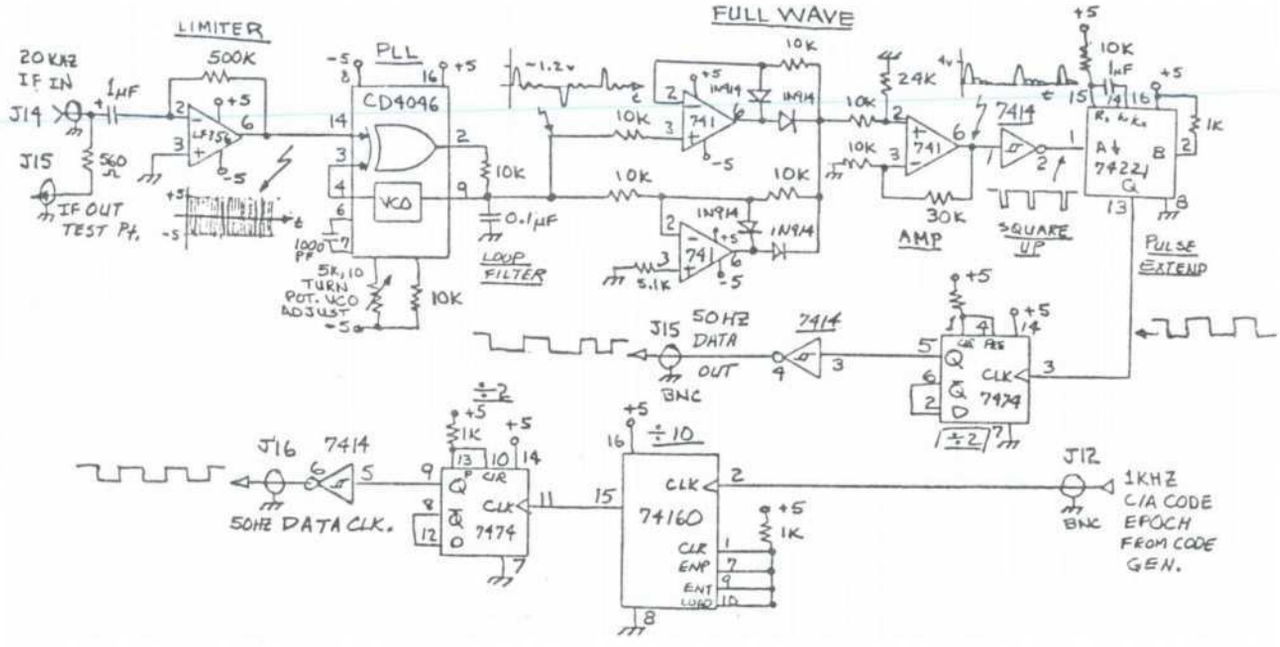


FIGURE 28

DATA DEMODULATOR / DATA CLOCK



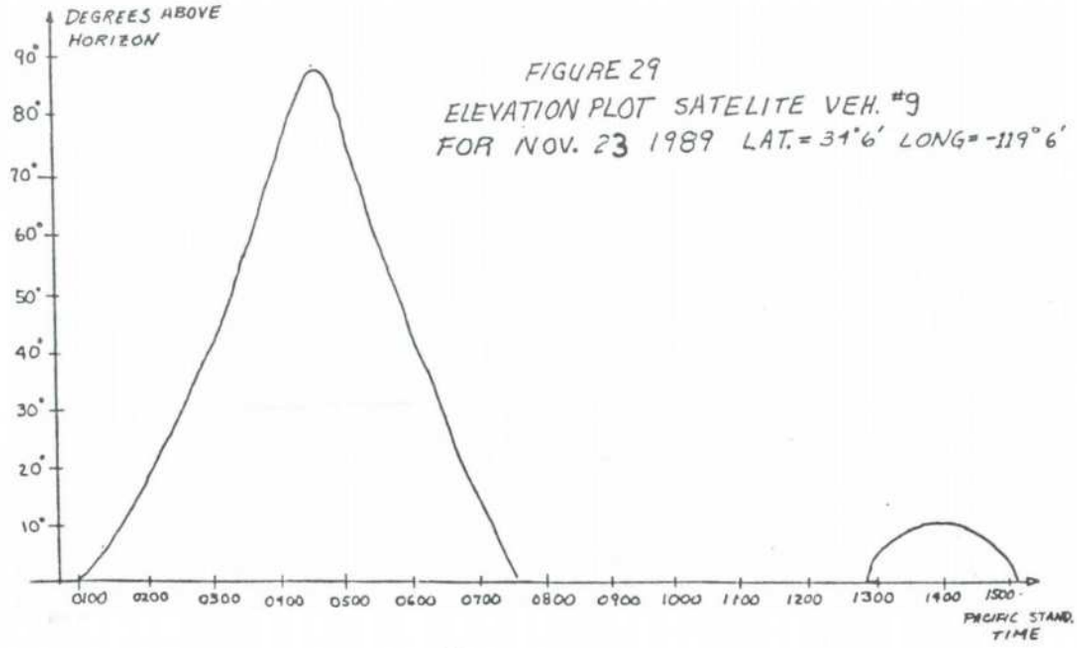


FIGURE 29

ELEVATION PLOT SATELITE VEH. #9

FOR NOV. 23 1989 LAT. = 31°6' LONG = -119°6'

FIGURE 30

DETECTED IF & DOPPLER VOLTAGES FOR NOV. 23, 1989
SAT. VECH. #9

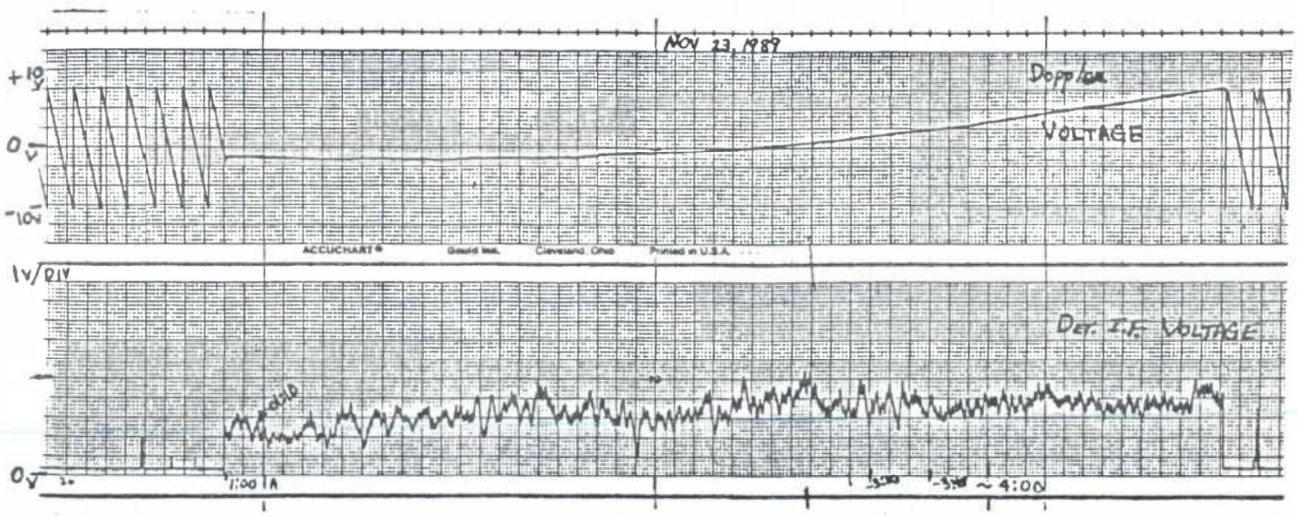


FIGURE 31

20 KHZ IF SPECTRUM FROM
SAT. VECH. #9, NOV 23, 1989; ~1400 P.S.T.
TAKEN AT J14 POINT - SEE FIG. 20

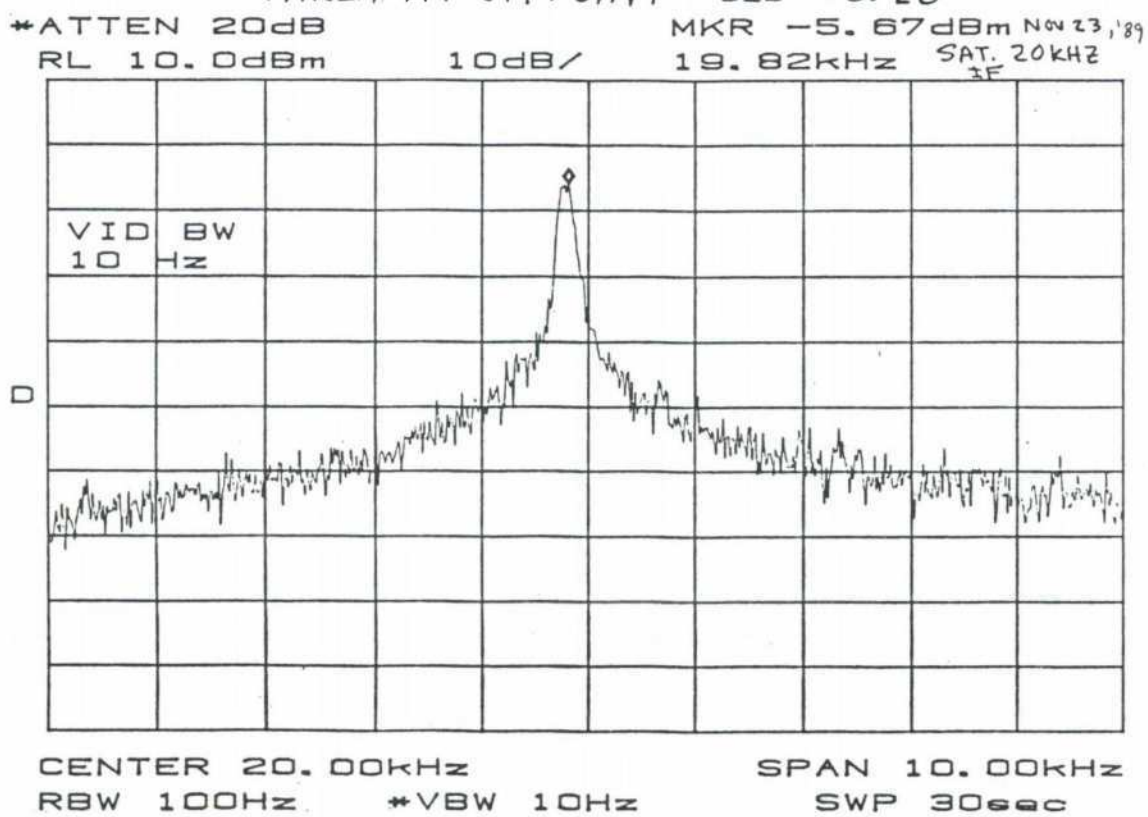


FIGURE 32
SAT. VECH. #9 DATA, NOV 23, 1989 (~0345 PST)

GPS02F Bit Count		PREAMBLE		FRAME I.D.	FRAME I.D.
13		10001011	11000000000111111101000	01111110011000010000001	11110000	1	6
73	300 BITS	10000000011010001000000011010111		10011001111000011000110111000000			
133		00111111000110100010011100011000		01001100110011110001001010101010			
193		11010010110001001111111111011101		11110100101000000011001011111101			
253		11111111100000000101110011101101		01001110101110001010101001110 0			
		PREAMBLE		FRAME I.D.			
313		10001011	11000000000111111101000	01111110011000011000101000110 0		2	5
373		00001011111111000001011001011 00		00001110100100100011111111111100			
433		01011000010011010100001110110 1		00000111000111011111100001101 11			
493		10001000111111111001110110100 11		11110000010101000101111000011 0			
553		00001101000000100010101000100 1		10100000001100101010100110100 0			
		PREAMBLE		FRAME ID.			
613		10001011	11000000000111111101000	01111110011000100000011 10100100		3	4
673		11111111011111111100001110110 0		01100110100111111110010000100 1			
733		00000000000111101110100100011 00		01011011011001111101000001010 1			
793		11011000110100011101010001100 1		00101001101100011000000011110 01			
853		00000000010001110000110000010 0		00001011111110010001111110101 00			
913		10001011	11000000000111111101000	01111110011000101001001 11000000		4	3
973		01110111001000000010000001101 1		11011111110111111101111110110 0			
1033		00100000001000000010000001001 1		11011111110111111101111110110 0			
1093		00100000001000000010000001001 1		11011111110111111101111110110 0			
1153		00100000001000000010000001001 1		11011111110111110101011101110 0			
1213		10001011	11000000000111111101000	01111110011000110000101 10010000		5	2
1273		01000000101010101010101010111 00		10101010101010101010101011110 0			
1333		10101010101010101010101011110 0		10101010101010101010101011110 0			
1393		10101010101010101010101011110 0		10101010101010101010101011110 0			
1453		10101010101010101010101011110 0		10101010101010101010101011100			
1513		10001011	11000000000111111101000	01111110011000111000001 01101100		1	6
1573		10000000011010001000000011010 11		10011001111000011000110111000000			
1633		00111111000110100010011100011 000		01001100110011110001001010101010			
1693		11010010110001001111111110111 01		11110100101000000011001011110 1			
1753		11111111000000001011100111010 1		01001110101110001010101001110 0			
1813		10001011	11000000000111111101000	01111110011001000000100 01111000		2	5
1873		00001011111111000001011001011 00		00001110100100100011111111111100			
1933		01011000010011010100001110110 1		00000111000111011111100001101 11			
1993		10001000111111111001110110100 11		11110000010101000101111000011 0			
2053		00001101000000100010101000100 1		10100000001100101010100110100 0			
2113		10001011	11000000000111111101000	01111110011001001000011 11110000		3	4
2173		11111111010111111111000011101 00		01100110100111111110010000100 1			
2233		00000000000111101110100100011 00		01011011011001111101000001010 1			
2293		11011000110100011101010001100 1		00101001101100011000000011110 01			
2353		00000000010001110000110000010 0		00001011111110010001111110101 00			
2413		10001011	11000000000111111101000	01111110011001010000100 01001100		4	3
2473		01111000001001000000010011100 0		11111110000000000010011000101 00			
2533		11110110111111111000000100011 10		11111111111111111100110100111 01			
2593		00000000000000000000000000101 01		11100010111101111111110111010 01			
2653		11111010111110110111111100011 00		00000111010101010101010000100 00			
2713		10001011	11000000000111111101000	01111110011001011000101 11000100		5	2
2773		01000000101010101010101010111 00		10101010101010101010101011110 0			
2833		10101010101010101010101011110 0		10101010101010101010101011110 0			
2893		10101010101010101010101011110 0		10101010101010101010101011110 0			
2953		10101010101010101010101011110 0		10101010101010101010101011110 0			

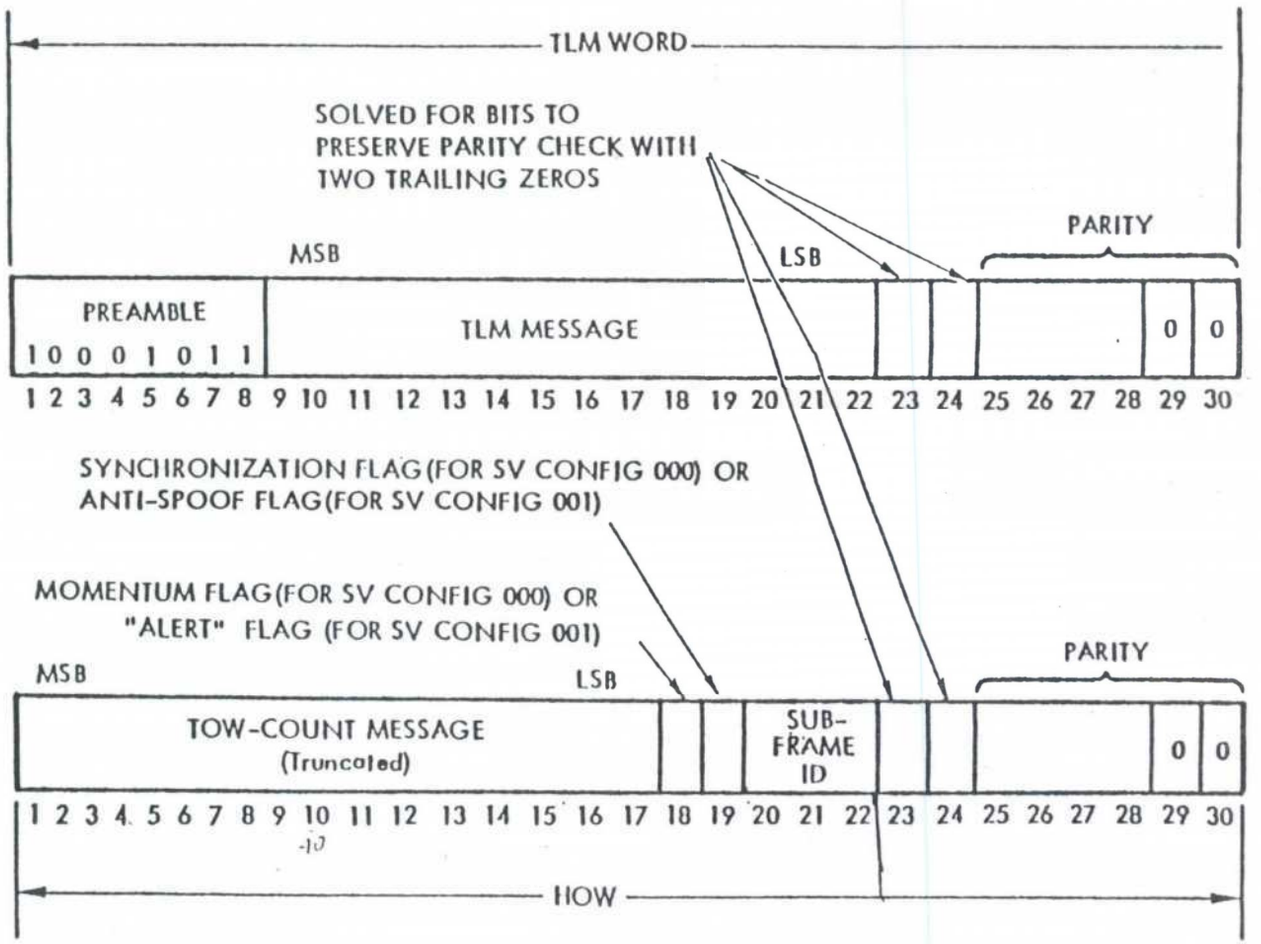


Figure 33. TLM and HOW Formats

FROM ICD-GPS-200

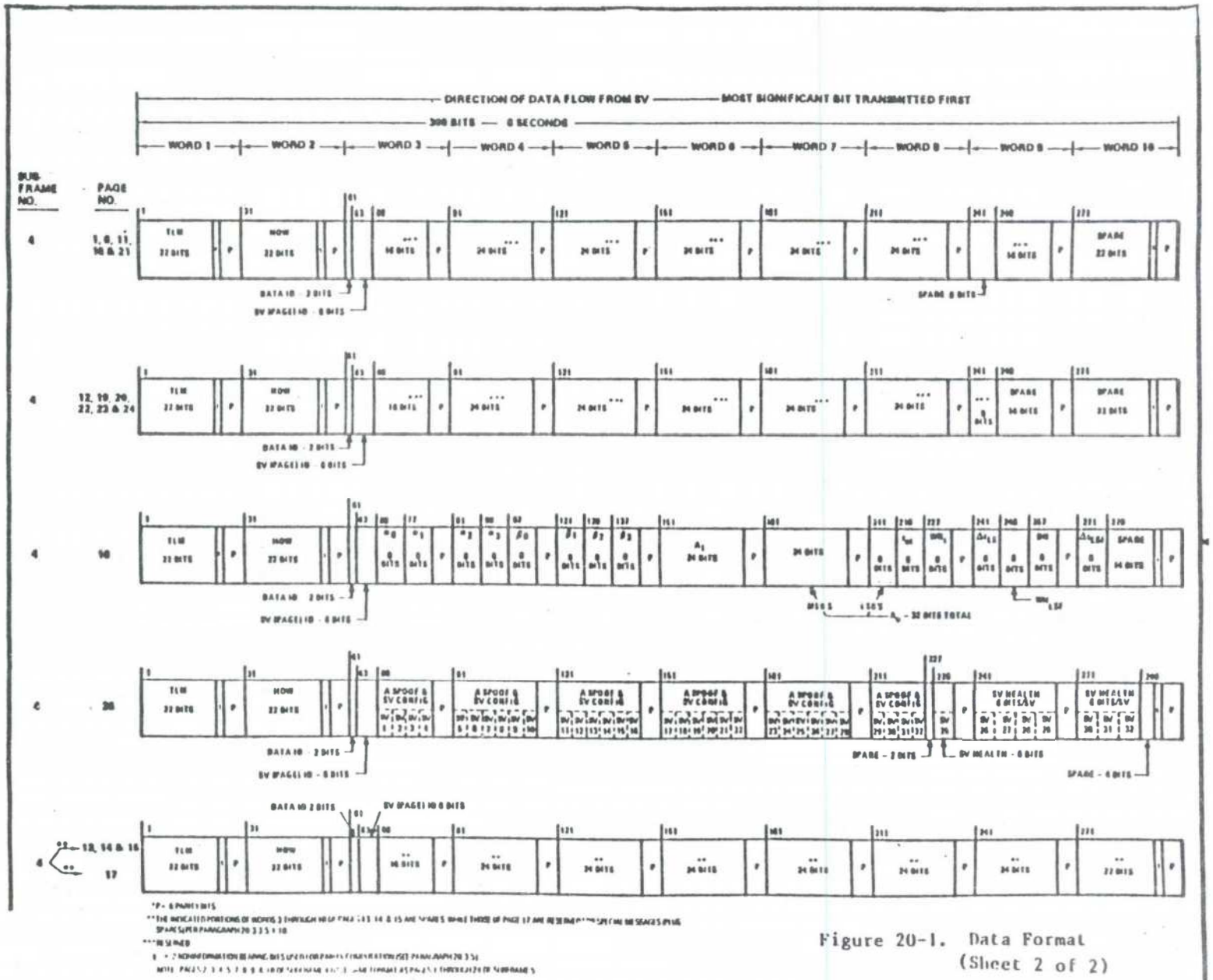


Figure 20-1. Data Format
(Sheet 2 of 2)

FIGURE 34A
DATA FRAME CONTENT CONT.
FROM ICD-GPS-200

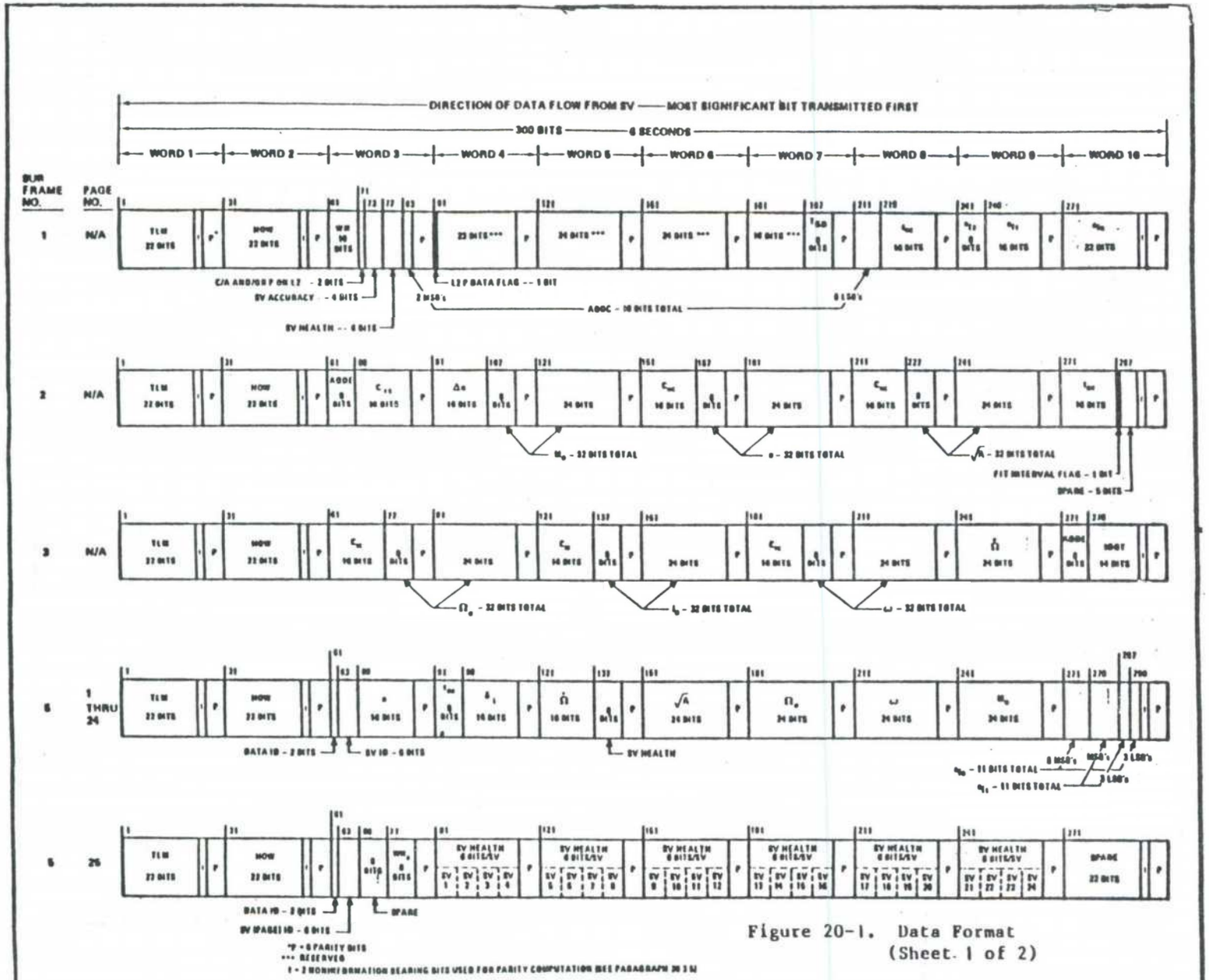


Figure 20-1. Data Format
(Sheet 1 of 2)

FIGURE 34B
DATA FRAME CONTENT FROM ICD-GPS-200

PHYSICAL LAYER CONSIDERATIONS IN BUILDING A HIGH SPEED AMATEUR RADIO NETWORK

Glenn Elmore N6GN
550 Willowside Rd.
Santa Rosa, CA 95401

ABSTRACT

The high information capacity required by an amateur high speed network requires optimum use of resources. Use of both directional antennas and the UHF and microwave bands is essential to obtain efficient use of hardware and spectrum resources. The use of shorter point-to-point links and small clusters of local users can achieve dramatic increases in user information throughput. Coordination and cooperation at all levels will be necessary to make a high speed amateur network a reality.

Amateur radio avoided dying in its infancy largely due to the establishment of a low speed digital network. The American Radio Relay League was founded to further this. Now we amateurs find ourselves entering the information age. I believe that if amateur radio is to continue in this age it must offer relevancy and that to do this amateurs must develop and implement a significant high speed information network; the alternative is for the hobby to wither and eventually die.

Although in the past radio amateurs have led the way into new technologies and operations, in recent times we have increasingly tended to adapt our operation and pursuits to existing technologies. Amateur packet radio, which has experienced very rapid growth in the last few years, is an example of this. Our interest in packet was stimulated by seeing similar communications within the industry and military complexes. The name given to the current link layer protocol, AX.25, is itself a variant of the name of a previous commercial protocol, X.25. Not only at lower layers do we see this borrowing of technology. The idea of a worldwide amateur BBS system and the greater dream for a digital amateur network have followed rather than led similar existing information services in the military and commercial sectors.

Certainly it is to be expected that more reuse of existing tools and methods will be required as our society and world get more complex. It is also true that insightful adaptation of methods and technologies often results in tremendous benefits. However, as we amateurs adapt our ways to meet the changing face of technology we need to examine the peculiarities of our applications, along with our strengths and weaknesses in order to achieve the most successful results.

I believe that if we are to succeed in developing and implementing a high speed amateur network that we must examine the fundamentals of communicating information by radio as well as our own resources and strengths and then design our network accordingly.

At the lowest ISO layers, physical and link, we have sought to implement packet communications by adapting existing hardware and protocols. Telephone modem hardware went into and is still inside most TNCs. Similarly, our link layer operations are tailored after the fashion of an IEEE 802 model. Both of these were originally intended for wire lines, a very different environment from amateur radio. I believe that many of the problems which amateur packet is now experiencing are traceable to this mismatch of solutions and environments.

A high speed amateur network requires a blend of two parts; high speed communication of information and wide area general access. High speed offers the opportunity for a great breadth and depth of applications. By

representing and transmitting a large amount of information digitally, the possibility for a wide range of applications exists.

A digital data stream can be used to represent voice, TV, FAX, as well as computer programs, files and data. Digital representation allows general transmission, storage and retrieval of this same data and also allows error detection and correction techniques to be used.

Along with this, a wide area network can allow amateurs to communicate and share resources in new ways. Such communication and shared resources could offer relevancy in the information age and rekindle the fundamental excitement and spirit with which the hobby began. The extent of possible applications of such a network applied to the diverse interests and pursuits of amateurs is truly staggering.

HIGH SPEED DATA

Neither of the above two ingredients has previously been seen in amateur radio. High data speed is necessary for both. Even a network providing low user speed requires high speed data if a large number of users are involved and all communication can not be carried out directly between end users. Any intermediate interconnection facilities become providers of a shared resource. If there is to be equitable sharing among many users then either the end users' data rate must be reduced for such communication or else these interconnections (possibly "backbones") must be capable of greater speeds in order to accommodate more than one user at a time.

A fundamental limitation to communication is noise. If this limitation were not present there would be no need for any particular transmit power, antenna gain or receiver bandwidth between two stations seeking to QSO. Whatever transmit power was recovered by the receiving antenna could simply be amplified as required to allow detection (recovery of transmitted information) to take place.

In actuality, signal power must be sufficient to allow separation of data from the noise power. Noise power at a receiver is

$$N = KT_eB$$

$K = 1.38 \times 10^{-23} \left(\frac{watts}{Hz\text{-}degrees}\right)$, Boltzmann's constant

T_e = effective system noise temperature, degrees Kelvin

B = Bandwidth, Hz

While good receiver design can reduce T_e, for terrestrial links the receive antenna is almost always receiving direct radiation from an earth which is approximately 290 degrees K. Combined with imperfections in the receive amplifiers, losses in the antenna system and QRM/QRN the effective unwanted signal (noise) level of the system is usually a little and sometimes a lot greater than this. Deep space links can effectively maintain lower system noise temperatures at the earth end but these are at present out of consideration for the bulk of radio amateur networking use and even if used, one end of such a link is earthbound and represents a high temperature noise source to the other.

Higher speed communication requires proportionately more signal power than lower speed. The Shannon limit[1], from information theory, sets the maximum channel capacity to be

$$C = B\log_2\left(\frac{S}{N}+1\right)$$

C = maximum channel error free data rate, bits/sec using a correct coding scheme

S = Signal power

This capacity is an ideal, common modulation methods may require signal power at least 10 dB greater than this. Also, unless signaling systems (modulation techniques) with a greater number of states M, (where M = bits/symbol = Bps/baud) are used then these faster systems always require greater bandwidth and incur an increased amount of system noise which must be overcome by similar increases in minimum recovered signal power.

The Nyquist theorem indicates that all information may be recovered by sampling a channel at a rate, R_b, no greater than

$R_b = 2B$, symbols per second (baud)

In this optimum case therefore $C = R_b M$ and equating with the Shannon limit above:

$$C = \frac{R_b}{2} \log_2 \left(\frac{S}{N} + 1 \right)$$

And the maximum number of useful bits/symbol, M_{max} is

$$M_{max} = \frac{\log_2 \left(\frac{S}{N} + 1 \right)}{2}$$

In high $\frac{S}{N}$ cases, the "excess" signal power can be used to purchase larger M and a resulting higher capacity at approximately $\log_2 \left(S \right)$. Using power to allow an increased bandwidth for a given $\frac{S}{N}$ is more effective since channel capacity then increase linearly instead of logarithmically.

However, whether or not complex signaling methods allowing more bps/baud are used or not, eventually additional bandwidth is required because there are practical limits to available recovered signal to noise ratio. This ultimately requires greater bandwidth as data rate is increased.

Since total information transfer is equal to an average transfer rate over a time interval, the product of rate and time, it can be seen that the total amount of information communicated is ultimately dependent upon the amount of energy transferred, the product of average power and time, between the transmitter and receiver. For the normal case of uniform mean noise, this energy must be enough greater than the noise energy in the same interval to allow successful data recovery.

The problem of efficient use of resources in designing and implementing an amateur high speed digital network then involves finding and utilizing the most efficient techniques for conveying (transmitting) this information-carrying energy. In an environment of limited resources of funding and spectrum, amateurs must use available resources optimumly if we are to build and operate a high speed network.

Signal Propagation by Radio as a Function of Carrier Frequency and Distance

As it is usually presented, the so-called pathloss equation shows what portion of the transmitted power gets to a distant location which is separated from it by a distance in free space. The assumption is that the distance is great enough that both transmitting and receiving antennas are in the far field regions of the other. The definition of far field distance, D_f is often considered to be:

$$D_f > 2\frac{D_a^2}{\lambda}$$

D_a = maximum antenna dimension, wavelengths

$$\lambda = wavelength = \frac{c}{Frequency}$$

In its most common form, the pathloss equation describes two isotropic antennas separated by a distance D. An isotropic antenna is an antenna which radiates uniformly in all directions. It is not physically realizable but is convenient for the sake of analysis. The portion of transmitted power recovered at the receiving end, L_{ii}, is

$$L_{ii} = \left(\frac{\lambda}{4\pi D}\right)^2$$

Given this representation, the amount of signal received decreases as either frequency or distance is increased.

A receiving antenna serves to intercept and recover a portion of the transmitted power. If the entire surface of a sphere of radius D were surrounded by perfect receiving antennas and if the outputs of all these antennas were totaled the sum would be the total transmitted power. No power is actually lost along a free space path. The amount of power a particular receive antenna recovers depends on how big an aperture or "bucket" it represents and the strength of the transmitted field at the antenna's position on the sphere.

To start to make this model more like a real-world situation we can substitute for the fictitious isotropic antenna a directional antenna which can actually be constructed. Except for excessive dissipative or matching loss, a directional antenna is by definition one which has gain. It gives gain because it causes power which the isotropic antenna would have spread evenly in all directions to be focused or concentrated in one or a few directions while reducing it in other directions. It essentially redirects power from some undesired directions to another desired direction. Antennas of higher gain have more of this focusing ability. The gain, G, of an ideal antenna may be stated in terms of its capture area or aperture, A,

$$G = \frac{4\pi A}{\lambda^2}$$

where A = antenna aperture, expressed in the same units as λ^2.

From the vantage point of the receiving antenna on the sphere it makes no difference whether the source is a 100W transmitter and an isotropic antenna or a 10W transmitter feeding a directional antenna with a gain of 10. The field strength at a receiving antenna located in the far field would be the same in either case.

Antenna effective aperture is a measure of the useful area of an antenna. Figure 1 shows the apertures of some familiar antennas.

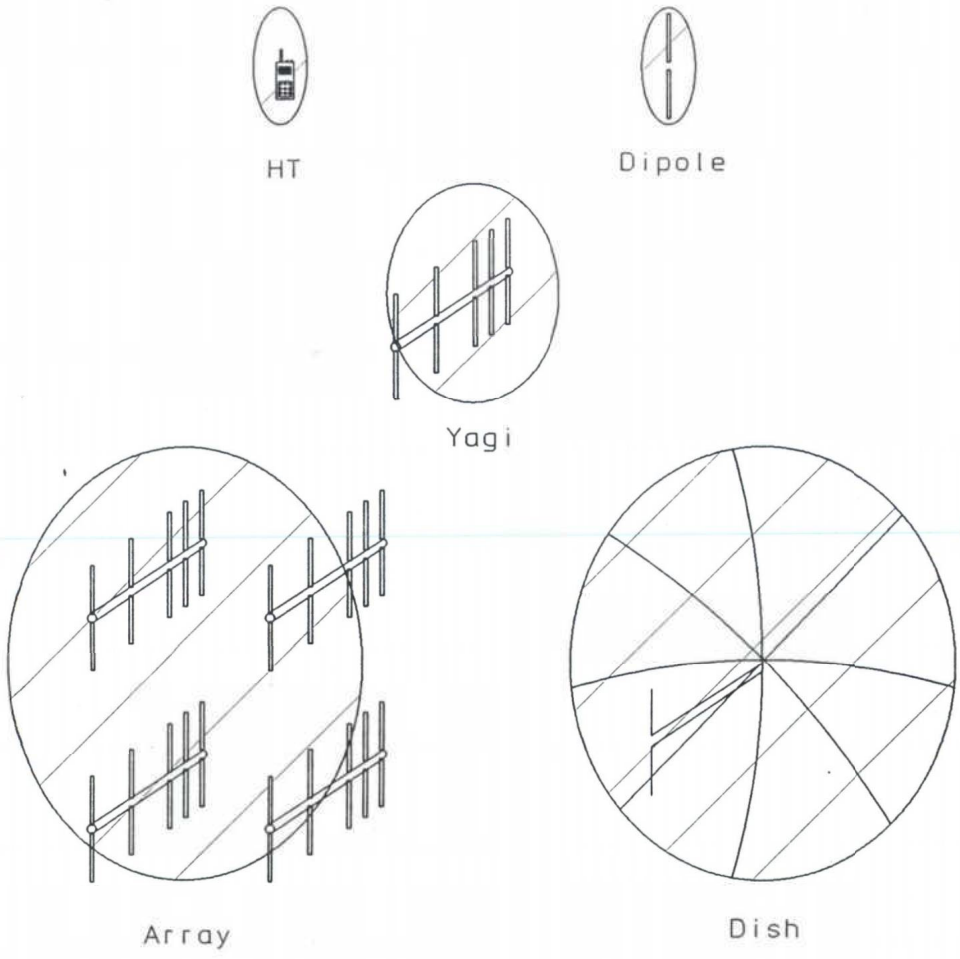


Figure 1. Relative Apertures Of Some Common Antennas

For simple single element antennas like a monopole or dipole, the aperture can be approximated by the area of a rectangle which is a half wavelength long and a quarter wavelength wide. It is not affected by the physical size of the conductor used to make the dipole. Also, even a shortened dipole or monopole, like the "rubber duck" antenna used on handheld radios, has almost the same aperture as that of a full-sized dipole. Consequently, this is the minimum aperture a real antenna can have.

Adding more elements, or electrical size, makes an antenna more complex and increases the aperture relative to a dipole or isotropic antenna. Antennas with more elements or electrical size have a relatively larger aperture on receive at the same time they have a more directional beam.

Now let's look at what happens to antenna aperture as frequency is changed.

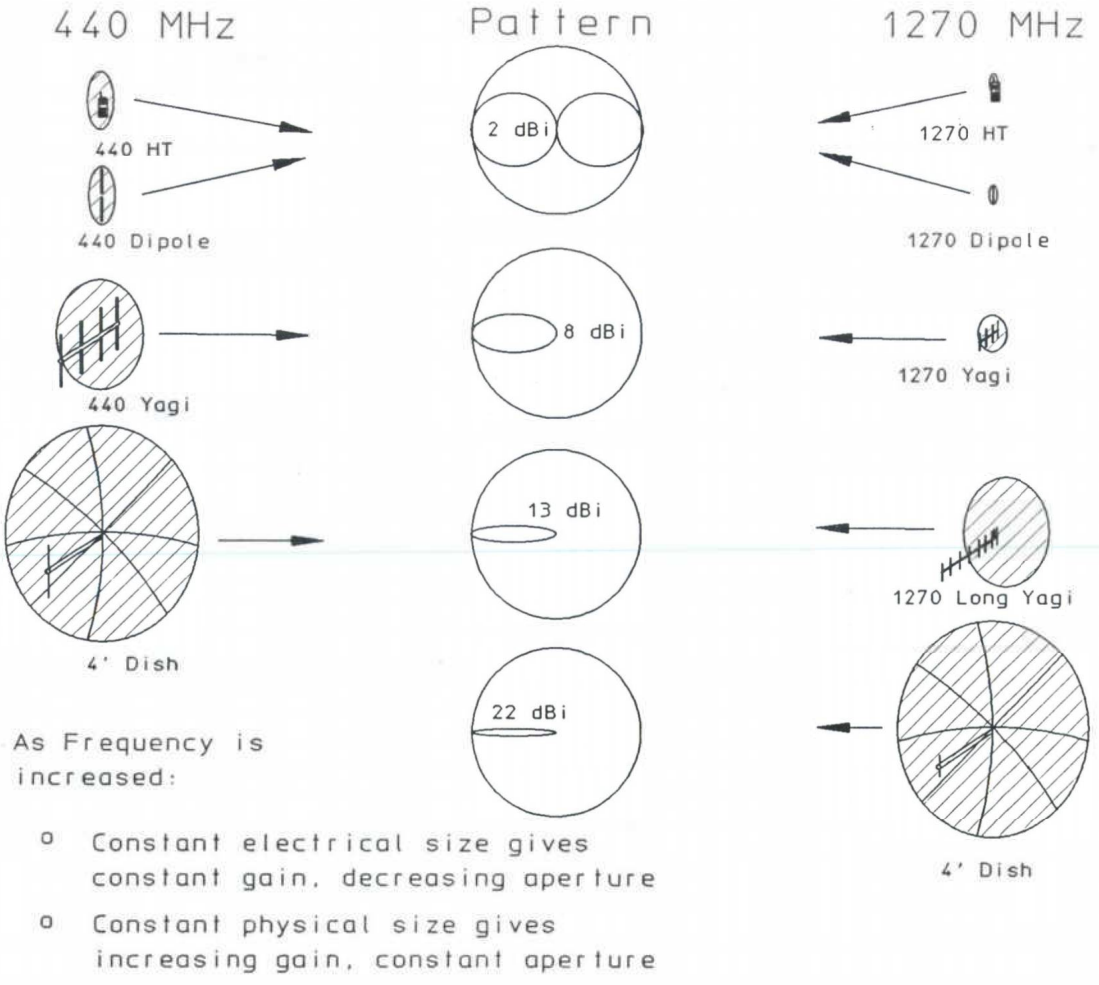


Figure 2. Antenna Apertures and Patterns Vs. Frequency

Here again are some common antennas and apertures shown to scale for two amateur bands. As the antenna electrical size or number of elements is increased apertures increase too but for a given electrical structure the higher frequency antennas start out with a smaller physical aperture because their wavelength is smaller.

As antennas much more complex than dipoles are considered, the aperture size is increasingly related to electrical antenna complexity. A 160M dipole may have more aperture or "intercept area" than a 150 foot parabolic reflector even though it is electrically a much simpler antenna. However, at frequencies where the dish antenna is electrically large, where it is at least 10 wavelengths in diameter, the physical aperture is relatively constant.

Notice the dish antenna near the bottom of Figure 2. It has the same physical size on each band, although its electrical size, measured in wavelengths, is about 3 times as large on 1270 MHz as it is on 440 MHz. It has about the same physical aperture on both bands although its gain, directivity and its ability to focus a transmitted signal is about 10 times greater at the higher frequency.

If we now substitute a directional receiving antenna of constant physical size and with gain, G_r, for the isotropic receiving antenna the portion of transmitter power transmitted between them, L_{id} is

$$L_{id} = G_r L_{ii} = \frac{K_{id}}{D^2}$$

where $K_{id} = \frac{A_r}{(4\pi)}$

K_{id} is a constant and the constant aperture antenna recovers the same amount of the transmitted signal irrespective of frequency. There is no longer any frequency dependence to the equation, only a distance dependence.

This arrangement of using a constant physical antenna size instead of a constant electrical antenna type makes a lot of sense in practice. Almost always the limitations to amateur antennas at the hamshack or at a high level site are in terms of antenna physical size rather than antenna electrical size. Antenna and tower wind loading, rotor capability and antenna size are constraints much more often than number of elements or antenna dimension measured in wavelengths.

We have now come to the point of realizing that there is nothing inherently wrong with increasing frequency when we are seeking to communicate between two different locations, that is, transmitted energy doesn't mysteriously evaporate in space. If we put up a given sized "collector" and transmit a given intensity in its direction the same amount of received signal may be recovered, no matter what frequency is used.

But we aren't quite done changing the antennas yet. In exactly the same way that the receive antenna size is more likely to be physically rather than electrically constrained, so is the transmit antenna size. We aren't limited to using a particular electrical size for transmitting. In fact we are likely to want to use the same antenna for both receiving and transmitting since most of our communications will need to be two-way.

In this third rendition of the equation we transmit as well as receive using an antenna of constant physical size. This result is also known as the Friis Transmission Formula[2].

$$L_{dd} = G_t L_{id} = \frac{A_r A_t}{(\lambda D)^2}$$

G_t = transmitting antenna gain

A_t = transmitting antenna effective aperture

Because the directivity and gain are now increasing with frequency the effective radiated power, ERP, is also increasing. The transmitter power is better focused to go only toward the receive antenna and not elsewhere, as frequency is increased. Once again there is a frequency dependence in the equation but this time instead of things getting worse as frequency is increased, as was the case with constant electrical antenna size, with constant physical antenna size the amount of transmitted signal reaching the receiver increases with increasing frequency. In fact, an increase in distance incurs no additional reduction in recovered power if frequency is increased by the same amount.

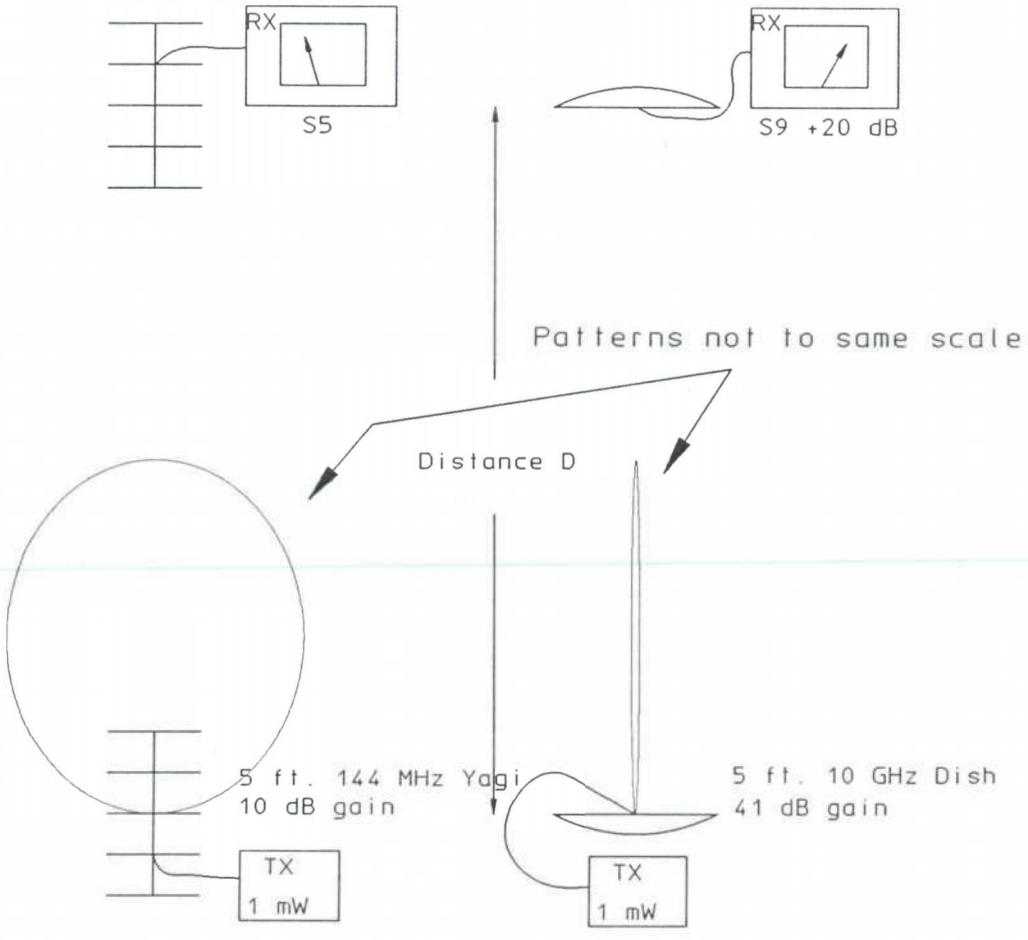


Figure 3. Practical Example Of Frequency/Directivity Improvement

As a practical example, Figure 3 shows that a transmitter using 5' antennas at 10 GHz can provide more than 1000 times as much signal to the receiver as the same transmit power and antenna size on 144 MHz. Or, as an alternative, 1000 times as much information could be transmitted over the 10GHz system as over the 2M one in the same time interval.

In summary, as frequency is increased when using constant physical antenna size for both receiving and transmitting the receive antenna has constant aperture and intercepts power within the same area but the transmitting antenna focuses the transmit power more resulting in more power recovered by the distant receiver.

Returning to the original goal of amateur communication of maximum "information carrying energy", we discover that to maximize "high-information-rate energy transfer", we must choose the highest available frequency where:

- Physical antenna size can be maintained
- Maximum ERP can be achieved. This is a trade-off of transmitter power expense for antenna gain.
- There is sufficient spectrum to support the signaling method and information rate
- Propagation over the desired path doesn't attenuate prohibitively more than the free-space line-of-sight (LOS) case

Higher frequency, shorter wavelength, point-to-point links are the most effective way to transfer information by radio and therefore offer the most attractive solution to the problem of communicating high volumes of information rapidly.

Within the context of current radio amateur resources, this is indeed fortunate. It is presently only the amateur microwave and millimeter bands which offer the possibility of highly directional antennas and have sufficient bandwidth to support high volumes of information transfer which will be necessary for a successful high speed digital amateur network.

RESOURCE SHARING

The goal of an amateur network is to provide information exchange among a large number of users; not just an optimimized, high rate information transfer across a single link. Such exchange needs to be performed within the bounds of amateur hardware and spectrum resources. Also these available resources and the resulting supported services must be equitably shared.

Having explored the requirement for highly focused beams for optimum information transfer over a single link, let's turn to the question of how point-to-point (PTP) physical and link layer implementations affect the sharing of resources compared with omnidirectional-to-omnidirectional (OTO) implementations.

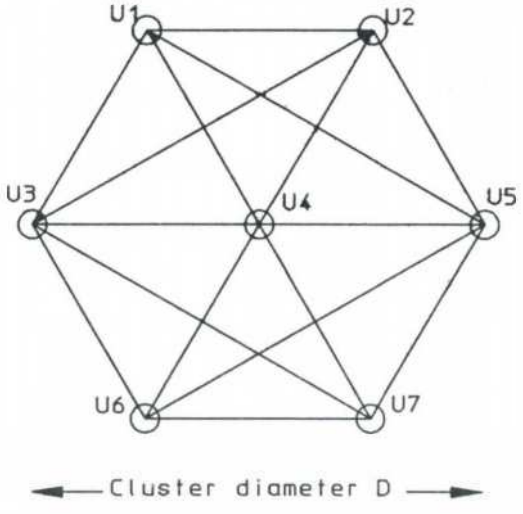


Figure 4. CSMA/OTO Cluster

Figure 4 shows an idealized representation of current amateur carrier sense multiple access (CSMA) packet radio. A number, U, of local users within radio range of each other are using omnidirectional antennas to receive one another's transmissions on a shared channel. If a "perfect" link layer protocol is in use, the channel capacity, C, may be thought of as being perfectly divided among these OTO users. If the channel is time shared each user may obtain the full channel data rate but only a portion of its information handling capacity. Each user can expect a share, R, of this capacity

$$R = \frac{C}{U}$$

$R =$ user throughput rate
$C =$ channel capacity
$U =$ number of users sharing the channel.

As anyone experienced with amateur packet on congested channels located in typical terrain knows, this is very definitely an ideal rate. The value of U is indeed variable at any particular time. The realities of hidden transmitters and packet collisions also degrade the ideal to an ALOHA case and rapidly reduce the throughput rate a great deal more.

When a station transmits in this environment, the channel effectively becomes unavailable to all other potential users. Transmitted power which is only being used to communicate with one other station (in a non-broadcast protocol) is causing the channel to be unusable by all other stations within radio interference range for the duration of the transmission. In addition, most of the transmitted power is going in directions other than that desired for that particular transmission. Consequently the data rate must be reduced so that the noise power in the occupied bandwidth is enough smaller than the recovered signal power to allow successful demodulation of the data. In this example, omnidirectionally transmitting the power is causing a double problem:

- Less than optimum use of both the channel and the stations' hardware
- Removal of the channel from use by other stations

Neither the information rate nor the resource sharing aspect of this implementation is optimum.

Fortunately, at the same time a PTP link more effectively transfers information it can also improve the resource sharing attributes of a network and improve the effective throughput rate available to end users. The directive nature of the antennas on a PTP link may reduce interference to other users at the same time it improves the performance of its own link relative to an OTO implementation.

An example may be useful to compare PTP link architecture with OTO.

Example Benefits of PTP Links

Figure 5 contrasts OTO communication with PTP communication. In the two cases stations are shown at the vertices and center of a hexagon. The users are assumed to be members of a cluster located in an environment with many other users present at constant population density.

Directional antennas with a gain of 20 dB, which corresponds to a 3 dB beamwidth of about 20 degrees, are chosen for the sake of the example. An operating frequency in the PTP case of 10 times that in the OTO case could provide this directivity while maintaining similar physical antenna sizes. In the PTP case there are six users and a server, added to properly route the information. The same shared channel width and antenna size is assumed. This might not be the case in practice since PTP dedicated hardware could easily have a dedicated channel too but the assumption allows fairer comparison with the OTO architecture. The hatched areas show the regions which are effectively "deallocated" during transmission. Any other potential network users within these regions can not expect to have the same channel fully available since transmissions by cluster members can make the channel unusable by another station using an omnidirectional receive antenna.

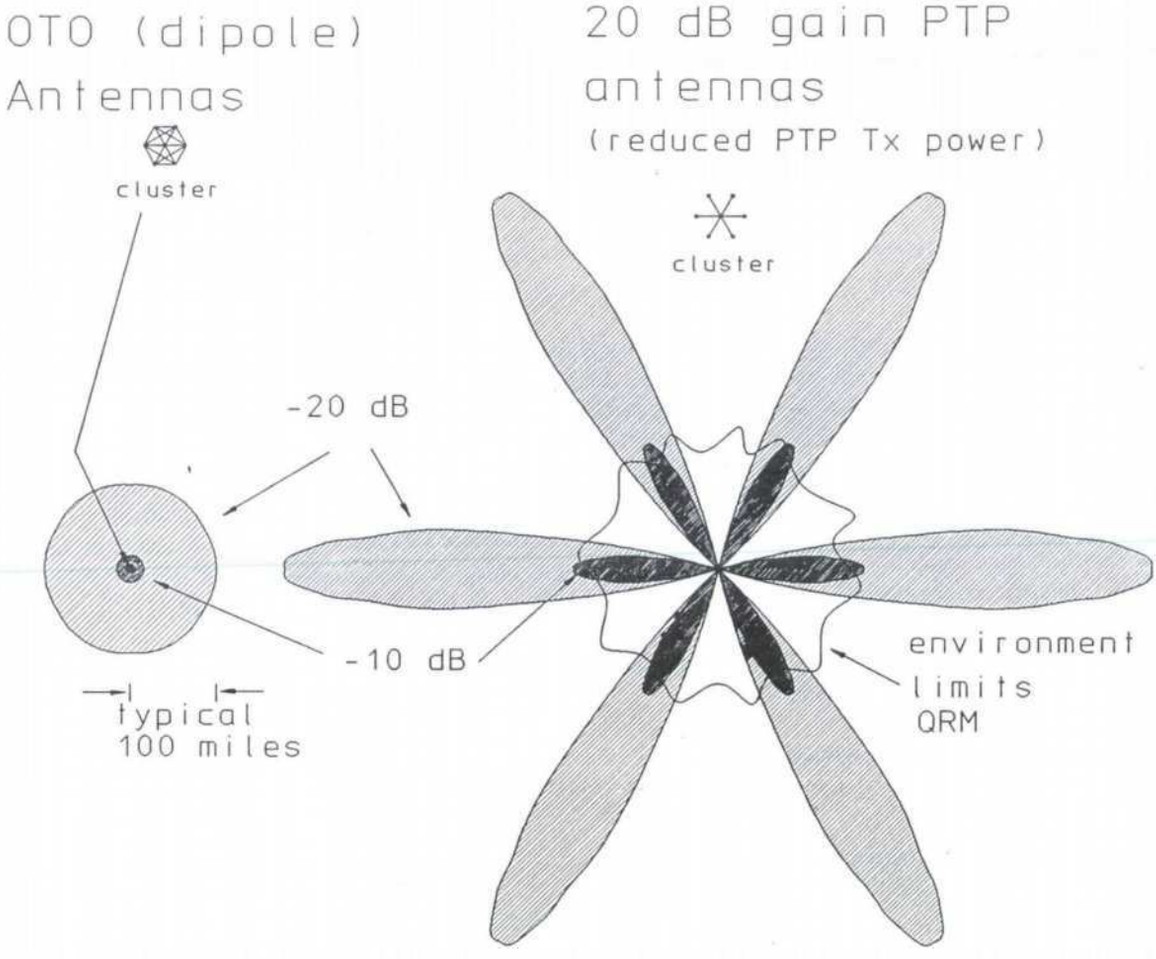


Figure 5. Comparison of OTO with PTP

The ends of these links are symmetric, the receive and the transmit antennas are the same. On receive the gain is a result of holding aperture constant as frequency is increased, on transmit this gain produces 100 times the ERP of the OTO case. This means that if transmitter power is held constant, 20 dB more signal power will be recovered. If a wider channel were used this could allow for an increase in the data rate of 100 times.

The PTP case does require the existence of a server, and a potential reduction of the throughput since the data must be sent twice, once to the server and again to the receiving user (our example uses only one shared channel and therefore no "cut-through"). But this is offset by the factor of two reduction in link path lengths; each user need only communicate with the server at the center instead of across the entire cluster diameter. This results in a net additional benefit. The potential improvement to a pair of users using PTP instead of OTO equipment is

100 (Tx antenna gain) X 4 (half pathlength compared to OTO)
X 1/2 (data must be sent twice) = 200 times

$C_p = 200C_o$ is available or $C_p = C_o$ with transmitters running 1/200th the output power.
where C_o = OTO channel capacity
and
C_p = PTP channel capacity

This improvement is at the cost of an additional radio and data processing hardware for the server. However

lower link budget margins are required to assure similar link up-time on the shorter path. In terms of energy and information transfer, the PTP case provides

$G_t + 6dB - 3dB = 23dB$ of improvement.

$G_t =$ transmit antenna gain

In terms of channel reuse, if the power is reduced the same user data rate may be supported with a fraction of the interference.

Antenna directivity and gain may be considered in terms of beamwidths. The gain of the directional transmit antenna increases because its beamwidths are decreasing. Use of a "square" antenna, one with similar dimensions in both vertical (elevation) and horizontal (azimuth) planes will produce similar beamwidths in these two planes. Thus a consequence of increased transmit antenna gain is a focusing of power in both planes. However if all the channel users in our example are located on the same horizontal plane only the narrowing of the beamwidth in azimuth serves to reduce interference. Increasing transmit antenna gain by G increases the ERP by this same amount but reduces the azimuthal beamwidth by only $\sqrt{G}$. The G times increase in ERP increases the range by $\sqrt{G}$ but at the same time the narrowing of the beamwidth reduces the subtended angle by $\sqrt{G}$. The net result is that the same total geographic area is affected in PTP as for the OTO case.

If user density, users/unit-area, is constant over the entire region, the number of users, U, affected by a transmission (to the -20 dB level) is

number of users impacted = Area x User density

$$U = \pi \left(10D\right)^2 \left(\frac{1user}{\left(\frac{D}{2}\right)^2} \right) = 1250 \text{ users (approximately)}$$

In practice solutions depend upon on the terrain and the relative elevations of the cluster members and the server. Also, in a real-world PTP situation, changes in the number and locations of cluster members may effectively deallocate a circular region. Coordination sufficient to maintain effective use of the "quiet" regions in between the petals of interference is probably not realistic. On a flat earth with all users in the same plane and only LOS paths the increased range of PTP could cause interference with any of G times as many users, effectively counteracting most of the PTP link performance gains with similar spectrum deallocation. However, propagation in real surroundings can help to limit this problem.

A realistic case might have a cluster diameter, $D = 10$ miles. If the server is located sufficiently elevated to be within LOS of all members it will probably be primarily the servers' transmitter which contributes to interference much beyond the cluster. This is because the cluster members locations are in general going to have absorptive ground clutter. Also for cases with narrow antenna beamwidths, perhaps gains above 35 dB, user antennas will be pointed upwards toward the server and the server's antennas will similarly be pointed downward. In one case transmitter power will radiate into space, in the other it will largely be absorbed by the earth. As long as the server isn't over-elevated, it is likely that at terrestrial distances greater than $10D = 100$ miles that attenuation will usually have increased by much more than the additional 20 dB LOS space loss because of reduced antenna gain at this vertical angle, scattering and absorption from distant clutter, and the earth's curvature. Energy which would have gone into interference on a flat earth winds up being absorbed or radiated into space where it ceases to cause a problem.

The likely result is that similar maximum areas will be affected in the two cases. If the PTP case causes interference out to, say, twice the range of the OTO case the number of affected users (to the -20 dB level again) is

$$U_p = \left(\frac{20D}{10D}\right)^2 U_o = 4U_o \text{ users}$$

U_o = number of users affected in OTO

U_p = number of users affected in PTP

However, if these potentially affected users are also using directional receiving antennas, perhaps as part of other similar small PTP clusters, virtual elimination of interference may be obtained resulting in potential for much better channel reuse and considerably greater improvement.

In this example then, the PTP architecture would provide user throughput rates, R_p of between

$$R_p = \frac{C_p}{4U_o} = 50\frac{C_o}{U_o} = 50R_o$$

for the case where the same channel must also be shared with other nearby users utilizing omnidirectional antennas to

$R_p = C_p = 200C_o$

if spectrum is reused with directional receive antennas and there is no competition for the channel.

Final user throughputs might then range from

$R_o = \frac{C_o}{1250}$, OTO user throughput rate for a congested OTO case to

$R_p = C_p = 200C_o$ rate for a non-contested PTP channel.

Thus the user throughput rate might be from one to two orders of magnitude improved in a PTP spectrum limited case when the channel is shared with OTO users. But if spectrum resources are not limited or if antenna directivity is great enough to allow complete frequency reuse, PTP users obtain the full $G_t + 6dB\text{-}3dB$ channel improvement and aren't required to share the channel as in the OTO/CSMA case. This could amount to a relative improvement of

$$R_p = C_p = 200C_o = 200 \text{x} 1250R_o = 250{,}000R_o,$$

more than five orders of magnitude while running the same transmitter power.

SOME PRACTICAL IMPLICATIONS OF A PTP NETWORK

As already indicated, in order to take best advantage of PTP hardware it is necessary to choose the highest available frequency where antenna size, ERP, bandwidth and propagation permit effective operation.

Antenna Size

To maintain the full $20\log\left(Frequency\right)$ advantage, the comparisons were made between antennas of constant physical aperture. The maximum acceptable size for an antenna will depend upon many things but a 145 MHz dipole or groundplane has an aperture of about $.25M^2$. This means that a properly constructed antenna with at least this much capture area will provide the expected improvement. Depending upon the antenna efficiency, the ratio of gain and directivity, the physical antenna size may have to be somewhat greater. An antenna of up to one meter on a side, seems acceptable for many uses.

A 1 meter dish can provide about 20 dBd (dB relative to a dipole) gain at 1250 MHz or 40 dBd at 10 GHz. To get this gain at 145 MHz would require a yagi nearly 10 meters long. At 1200 MHz and below yagis, arrays of yagis and collinear antennas are convenient. A very inexpensive conical approximation of a parabola can be made for 1200 MHz from "rabbit wire" having nearly 20 dB gain. At the high end of the amateur microwave

range solid reflectors are probably most suitable. Surface accuracy which generally needs to be better than $\frac{\lambda}{8}$, about 1/16 inch at 24 GHz, is commercially available in inexpensive dishes.

An upper limit on antenna size relates to propagation as described below.

Transmitter Power and ERP

To about the 100 milliwatt level, the cost of transmitter power is roughly constant through 10 GHz. To the 10 watt level it is approximately constant through 1300 MHz. Since using the minimum necessary power improves the network resource sharing, (not to mention being mandated by FCC regulation), it is desirable to use antenna gain, and higher frequencies rather than transmitter power to obtain the channel quality necessary for a given throughput rate. As other articles have already shown, transmit power of a few milliwatts is sufficient to communicate several Mbps. over a distance of 40 miles with 2 foot antennas[3].

Available Spectrum

In general, available bandwidth increases with increasing frequency. In most areas, the amateur bands above 450 MHz are underutilized. The largest contiguous amateur microwave band is presently at 10 GHz where 500 MHz is available. This is an extremely valuable resource which should be used.
Present FCC regulations also promote use of higher frequencies for higher baud amateur communication.

Propagation

For effective use of resources it is necessary to have high quality, LOS paths. When none is available it will be desirable to break the path into multiple shorter paths. As shown in the PTP example, shorter links can produce a net improvement in performance. Presently many packet paths are not LOS. At vhf and lower frequencies, diffraction allows for some "bending" over obstacles and terrain. This is very expensive from a link budget point of view and in part accounts for much of the mediocre operation of hardware which should otherwise be capable of much better performance (typical 10-100 watt ERP and $>.25M^2$ apertures). Even through the 24 GHz band, atmospheric attenuation is probably not a significant problem over the shorter path lengths which are desirable.

In situations where longer paths are unavoidable, perhaps on some backbone link connections, data rate may have to be decreased to ensure high probability of link function. The chief problem with longer paths is not the additional $20\log\left(D\right)$ path loss so much as the variability in propagation through the troposphere. In well stirred air (as in parts of the Rocky Mountains) this is less of a problem but some areas, particularly near large bodies of water, exhibit great variability due to moist air masses. These variations on longer paths may make vhf-microwave DX interesting but they can require a lot of excess system margin to guarantee link availability.

These variations and the beam "de-steering" that tropospheric refraction can produce sets an upper limit on antenna gain. When the gain gets too high and the corresponding beamwidth gets too narrow the beam can be deflected from the intended direction. This sets an upper limit to available performance on a path. To achieve higher data rates the longer path must be broken up into multiple shorter paths which exhibit a smaller amount of deflection.

Other Considerations

In addition to the above constraints there are many other higher level issues which must be considered when designing and implementing physical and link layer network components to make optimum use of amateur resources.

From an efficiency point of view both mean square path length and total network path length need to be minimized. To do this, clusters need to be limited to a small number of members. However, operation of a

local cluster may require the resources and cooperation of several individuals in the same way that local NBFM repeaters are constructed and maintained. A network of small high performance clusters requires more servers and more expensive higher performance hardware. Larger clusters require faster servers capable of processing many simultaneous higher layer communications.

Radio amateurs may be uniquely positioned to implement and maintain a wide area network. The physically distributed nature of the hobby's membership combined with radio communications skills and access to higher level sites in many locales may prove invaluable in creating such a network. To encourage amateurs with these resources to help construct a network may require some network applications which are particularly attractive to their interests. Applications supporting digital voice QSO, linking with other remote users and control and monitoring of existing analog hardware at high level sites may be attractive and help enlist their efforts in supporting a backbone of high speed hardware at high level sites.

For some amateurs, location may be such that no cluster server is available or easily constructed with resources at hand. In these instances it may be necessary that another amateur within LOS provide equipment full time to allow access to the network.

In general, higher user throughput data rates will require higher performance links and more coordination. Hardware, associated protocols and available user speeds may have to gradually migrate toward the ideal of small very high performance clusters. There will need to be ongoing access to such a network by entry level, lower speed users.

Perhaps above all else, the distributed and shared nature of a network demands organization and cooperation among amateurs. This may in fact be the most difficult task.

Local organization is required to install and maintain clusters and backbone connections. Between local clusters frequency coordination must be sufficient to allow good spectrum reuse, particularly as performance and channel bandwidths increase. Such coordination deals with a problem similar to the mapmaker's multi-colored map problem; how to assign a limited number of colors (channels) to a large number of areas (countries or states) so that adjacent domains do not have the same color.

Protocols which provide dynamic network routing without user involvement or knowledge need to be developed to allow for regional communication. Similarly national and international organization needs to oversee long distance connectivity. Satellites and other long haul links need to be developed and efficiently administered. It will likely take the combined resources of amateur radio to do this.

SUMMARY

The hobby of amateur radio nearly died at its inception. It was relegated to shorter "useless" wavelengths and left to die. Fortunately for amateurs the value of this spectrum combined with group cooperation and networking established the hobby in the face of this opposition. If we are to continue as radio amateurs, the hobby must again achieve relevance within our culture through efficient use of our resources. In many ways we are uniquely positioned to do this but successful execution will demand attention to methods and a great deal of cooperation. The potential result is the birth of truly exciting new aspects to ham radio and the continuation of the king of hobbies.

REFERENCES

1. W. Lindsy and M. Simon, "Telecommunication Systems Engineering", Prentice-Hall 1973

2. J. Kraus, "Antennas", McGraw-Hill Electrical and Electronic Engineering Series, 1950

3. G. Elmore and K. Rowett, "Inexpensive Multi-Megabuad Microwave Data Link", Ham Radio Magazine, December 1989, pp 9-29

Hubmaster: Cluster-Based Access to High-Speed Networks

Glenn Elmore N6GN
Kevin Rowett N6RCE
Ed Satterthwaite N6PLO

Abstract

This paper describes the link-level design of Hubmaster, a packet radio network that we are currently building. Hubmaster is intended to offer medium-speed network access to collections of end users and to serve as a stepping-stone to higher-speed networks. Users are organized into local clusters. Each cluster consists of a centrally located hub and a dynamically formed group of secondaries. The hub time-multiplexes a single channel by polling the secondaries. Our initial design will provide 256 kbit/s links and operate on the 33 or 23 cm band.

1. Introduction and Summary

Amateur access to high-speed packet radio networks will enable a number of intriguing applications [1]. Several of us in northern California hope to learn how to organize and implement such networks by building prototypes and trying to use them. We are as much interested in overall system design as in the details of the particular components. Our goal is to accommodate varying levels of link performance and to provide cost-effective hardware at each level of performance. In particular, we believe that high data rates must be available to end users, not reserved for backbones and the like.

This paper addresses what we consider to be a medium-performance level, with maximum data rates of 100 kbit/s to 1 Mbit/s. It describes the link-level design of a local radio cluster intended to offer network access to end users. Clusters will be connected as transparently as possible by point-to-point links using, e.g., microwave technology that has already been demonstrated [3, 4]. The design of the clusters borrows ideas from local area network (LAN) technology, but users should not view a cluster as just a LAN. Intercluster links are essential parts of our overall architecture, but they are not discussed in detail here.

Each cluster consists of a hub and a dynamically formed collection of secondary stations. All packet exchanges go through the hub and take place on a single, shared RF channel. It is not necessary for secondaries to communicate directly. The hub communicates with one secondary at a time over a half-duplex link. A polling discipline administered by the hub is used to time-multiplex use of these links in a collision-free manner. The polling list is formed dynamically and the collection of active users can change over time. The hub also serves as a store-and-forward switch. It accepts packets from each secondary and subsequently delivers them to another secondary, which might represent a local user, a link to another cluster or a gateway to another network.

We believe that our design will provide a useful increment of performance in a way that is cost-effective, flexible and easy to manage We have not yet tested our ideas; implementation of the hardware and software pieces is still in progress, and no doubt some of the details will change as the design continues to evolve. Thus this paper should be considered an interim report and a request for comments.

2. What Are We Building?

Our initial implementation will provide 256 kbit/s links on the 33 cm band (904/916 MHz). Because of the somewhat dubious future of that

band, we are already planning 23 cm versions of the RF hardware.

We envision a typical cluster diameter of 10 to 20 miles. Coverage of a single cluster is intentionally limited so that bit error rates can be kept low and latencies due to propagation delays are small. The number of active users supported by a cluster at any instant depends upon the traffic patterns and level of service required. We estimate that a dozen or so active users can be accommodated easily. Adding users decreases the effective data rate for each user and increases the latency but has little effect on channel throughput. We have tried to keep the cost of the shared components of a cluster as low as possible, so that growth in the user community can be accommodated by forming new clusters, not by degrading service.

At the logical and RF center of each cluster is a hub. The hub is a shared resource, somewhat comparable to a shared FM repeater. The hub consists of some digital hardware for polling and switching, a radio with an integrated modem, and an omnidirectional antenna.

A dedicated PC with sufficiently fast serial ports would be adequate for the digital end, but we are building a leaner and more specialized board called Mundane I/O or MIO. MIO is a simplification of the previously reported Awesome I/O design [2]. It consists of an NEC V40 microprocessor, program and buffer memory, and a pair of Zilog 85C30 serial channel controllers (SCCs), both DMA-driven. The V40 is code-compatible with the 8086/8; thus inexpensive and widely available software can be used for program development. The V40 is also CMOS, for low power, and highly integrated, for a reduced parts count. MIO can be used either in an ISA expansion slot (IBM PC, XT or AT compatible) or in a stand-alone configuration. In the former, the PC and the V40 communicate through a shared window into MIO memory. In the latter, an MIO can function as a self-contained hub or it can be connected to a non-PC through another SCC port, e.g., to a Macintosh via LocalTalk.

The 904 MHz radios are crystal controlled and provide a maximum output power of about 10 watts. Modulation is FSK; at 256 kbaud, the RF channel bandwidth (sidebands down 70 dB or more) is 2 MHz. The receiver is able to recover data at signal levels as low as -90 dBm. The radios also feature a turnaround time of less than 40 μsec (transmit to receive). They are built on a pair of PC boards and fit into a 5" x 7" enclosure. Further details of the prototypes are reported in [6].

The hub must be able to communicate with all the secondaries and thus uses an omnidirectional antenna. This can be a gain antenna in the vertical plane, and we expect to use a collinear array to achieve a "pancake" pattern with 10 to 12 dB gain. The hub antenna should be elevated just enough to have paths that are line-of-sight or nearly so to all secondaries. Too little elevation produces path losses that reduce system margin; too much interferes with channel reuse among multiple clusters.

The hardware associated with a secondary can be more modest, although as an expedient we will use copies of the hub hardware in our initial tests. One critical difference is the antenna; a secondary is equipped with an antenna that is directional in both axes toward the hub. We plan to use 15 element Yagis with 4.2 λ booms, constructed to NBS designs [7], with a gain of approximately 14 dBd and a half-power beam width of about 30°. There is enough system margin in the design to allow secondaries to operate at lower power levels; we expect 100 mW to be adequate for line-of-sight paths. We have ideas for even simpler digital and RF hardware for the secondaries, but its development currently has low priority.

In operation, the hub controls use of the RF channel by roll-call polling. The polling protocol is loosely based on HDLC NRM (see, e.g., [10, pp. 254-257]). For each secondary in turn, the hub sends any accumulated traffic for that secondary, followed by an invitation to transmit. The secondary, and only the selected

secondary, responds with any waiting packet traffic. The packets are received by the hub, stored, and forwarded to the local destination the next time that destination is polled. Since a secondary may transmit only upon invitation, no packets are lost to collisions.

For greater efficiency, the polling exchanges are piggybacked on the data packets when possible. Also, to keep intracluster routing and switching overheads to a minimum, special short addresses are used within the cluster as described below. The polling list is formed dynamically. The hub periodically broadcasts an invitation for new stations to join; collisions and conflicts are resolved by the usual backoff algorithms. Secondaries are dropped from the polling list by a time-out mechanism.

The hub is linked to hubs in other clusters via one or more high-speed channels. Selected clusters can share a local address space. Traffic between such clusters is simply forwarded to the other cluster by the hub. For other traffic, software in the hub must provide bridging. Any secondary can also provide a bridge or gateway to other networks. In either case, links leading out of the cluster can operate in parallel with the local polling and forwarding as long as any RF channels are adequately isolated and the nodes have sufficient computing power.

3. Why Are We Doing It This Way?

Packet radio using CSMA protocols on the VHF and UHF bands is well-established. The data rates (1.2 to 100 kbit/s) are adequate for many purposes. We do not expect it to disappear, and we plan to offer access to and from the AX.25 network through gateways. One of the greatest strengths of the existing network is also one of its greatest weaknesses — each station is independent and very little coordination or centralized network management is required. Also, the entry cost is very low.

At the other end of the performance scale, existing technology can provide data rates of 1 to at least 10 Mbit/s. The bandwidths to support such data rates are available only at UHF and microwave frequencies. In a companion paper, one of us (N6GN) argues that point-to-point links using high gain antennas are not only necessary but also very attractive at these frequencies [5]. Another of us (N6PLO) has been involved in the development of a high-performance experimental LAN based on full-duplex point-to-point links [9].

We believe that an ultimate packet radio network will be constructed primarily from such links. In the short term, however, there seem to be some practical problems. Providing a digital interface capable of sustained operation at these rates is still expensive; indeed, many personal computers currently in use don't have the memory and I/O bandwidths to benefit significantly from such data rates. The RF circuitry to support full-duplex operation is also more complicated and expensive; in addition, more spectrum and coordination are required, since each such link requires a pair of RF channels.

Finally, there are a number of problems created by the use of point-to-point links and highly directional antennas. The network configuration is less flexible; dedicated antennas and radio hardware are needed for each link, and the end points of new links must be negotiated and coordinated. Also, configuring point-to-point links into a useful network is a challenging problem. Unless a special and highly regular topology such as a ring is used (probably impractical in amateur radio), there must be switching nodes within the network that can cooperate to forward a packet from any source to any destination. Such switches are endpoints of multiple, simultaneously active links, and their antenna systems thus can become unwieldy for amateur installations.

The design presented in this paper is a compromise. It offers an intermediate level of performance at a lower cost and in a more flexible way. Logically, the intracluster links are point-to-point (except during well defined

intervals when new stations are contending to join the cluster), so the problems of organizing such links into a useful network can be addressed and potential solutions evaluated. In addition, one end of each link, the secondary, does benefit from the use of gain antennas. Because the hub is omnidirectional, however, its antenna system is simple, and the set of possible secondaries is limited only by path length. The latter is an advantage for portable and emergency operation. Although resources are used less efficiently, nothing breaks even if it is necessary to operate the secondary with an omni antenna in such circumstances.

A companion paper [5] shows that path loss is independent of frequency in the point-to-omni configuration of a cluster if the physical antenna sizes at the secondaries remain constant. This might argue for using the lowest frequency at which the desired bandwidth is available, since the cost of generating and feeding RF power generally increases with frequency. Actually, by using a collinear array of constant aperture at the hub, we can make limited use of the decrease in path loss with increasing frequency. Construction of such antennas with satisfactory patterns becomes relatively more difficult above 1300 MHz, as does the generation of power at levels above 100 mW. Thus our initial experiments with a cluster configuration use the 33 and 23 cm bands.

We have been influenced in our choice of bands by other pragmatic considerations as well. Both bands are underutilized and need more activity. The membership of our group is currently too scattered to be connected effectively by short direct microwave links. The use of a few longer, non-line-of-sight paths helps considerably. Finally, we were able to obtain some relatively inexpensive surplus parts suitable for use at 900 MHz.

To control channel access, we have chosen to use polling. Since the secondaries cannot in general hear one another, conventional CSMA fails. Performance of pure Aloha, into which it would degenerate, is unacceptable. One alternative that retains CSMA operation would use a repeater at the hub, but the required RF hardware would be more difficult to design and, in our judgment, considerably more expensive. As a repeater, the hub must operate full-duplex. The secondaries must at least transmit and receive on separate frequencies, and they must also be full-duplex if CSMA/CD is desired.

Polling is not free. The polling exchange is an overhead that reduces effective channel capacity and, perhaps more seriously, increases latency. Performance is worst when there are large numbers of secondaries on the polling roll but few of them are offering traffic. This is one reason we believe that the size of the polling list must be limited. With a single channel and half-duplex links, the maximum throughput is half the channel data rate, even if there are no other contenders for the channel. On the other hand, polling has its charms. The effective data rate of the channel actually increases with load and, because of more effective piggybacking, with the number of stations offering traffic. We view polling as an expedient offering acceptable trade-offs at the level of cost and performance that we currently wish to explore, not as an ultimate solution.

4. How Does It Work?

This section explains part of our design in more detail. It focuses on the link level. Many of the design decisions and algorithms are arbitrary; it is easy to think of variants that might have more desirable behavior in one respect or another. We have tried to keep our initial design as simple as possible while providing adequate performance. One reason we want to build a network is to explore just such alternatives.

A. Packet Formats

Packets generally follow the HDLC (ANSI X3.66) frame format. We wanted to use an established framing convention that is compatible with commercial LSI circuits, such

as the popular Zilog 8530 SCC. This hardware provides satisfactory frame delimiting and data transparancy (bit-stuffing). It can also be programmed to do address-based packet filtering in a way that we exploit.

Our frames have the following format:

flag-DA-SA-C-DUA-SUA-I-FCS-flag

Flags (binary 01111110) delimit the packet. Zero stuffing is used to insure that data bytes will not be recognized as flags.

DA and SA, the so-called cluster addresses, are encoded as single bytes. Normally, these are the cluster-relative addresses of the destination and source secondaries respectively. The exact set of values and their intended uses are described in a following section. The SUA and DUA fields contain the call sign and chosen SSID of the source station and the destination station. Each field is eight bytes in length. These are included to satisfy FCC requirements. At the link level, they also uniquely identify the source and destination and serve as cluster-independent addresses.

The C field serves two purposes. It includes a Direction bit and a Poll/Final (P/F) bit. These are used to piggyback polling and completion messages onto other frames. The Direction bit is needed to resolve ambiguities that could otherwise arise when two secondaries can hear each other directly. It is set to one in packets from secondaries to the hub. The rest of the field is a packet type code. Some packet types are reserved for the protocol used to do dynamic cluster configuration; the others are not distinguished at the link level.

The I field is the information field. Its contents are arbitrary. The maximum length of an I field is 1500 bytes. This allows an Ethernet to Hubmaster gateway without packet fragmentation. The FCS field is the 16-bit CCITT V.41 CRC used to check for corrupted bits.

B. Polling Operation

The hub maintains a list of active stations and sequentially considers each one in turn. If the hub has any packets for a secondary, it transmits those packets first and sets the P/F bit of the last packet in the sequence. Otherwise, it sends a minimum length packet with the P/F bit set. This grants control of the RF channel to the secondary, which then sends any waiting traffic to the hub. The secondary marks the end of its traffic by setting the P/F bit in the last packet. The master stores any information frames that it receives and goes on to poll the next secondary. If the traffic is destined for another station in the same cluster, it is delivered as part of polling that secondary. Traffic for an intercluster link is delivered to a process within the hub for forwarding.

The protocol does not limit the number of frames sent in response to a poll. Since the hub can always be jammed by a malfunctioning or malicious station, there is no way to enforce such a limit. The hub can impose a certain amount of flow control by withholding the P/F bit when its buffers are too full. We expect protocols at higher levels to provide the primary flow control.

Similarly, our protocol makes no provision for packet acknowledgment. We believe that acknowledgment, if required at all, is an end-to-end function [8]. Since we are specifically designing the intracluster links for low bit error rates, we do not implement HDLC's sliding window protocol or anything similar at the link level. Packets that arrive with incorrect CRCs are tallied and discarded; the tallies are used to identify and report marginal links.

Delay	Time (μsec)	Notes
transmitter turn-on	10	
preamble transmission	250	8 bytes preceding the opening flag
packet transmission	719 + 31.25N	23 bytes of mandatory packet fields; N = data bytes in the I field
propagation	3.34R	R = cluster radius (km)

Table 1: Delay Times

Polling and packet overheads determine the performance of this protocol. Table 1 shows a breakdown of the time to transfer a single packet. The preamble is required for the clock in the receiver to lock to the received data stream. We have allowed 8 bytes for the 85C30 SCC (NRZI encoding), which we believe to be conservative. The propagation delay is not an important contributor at the data rates and distances being proposed here. We use the delay for a path of 15 km (9.4 miles) in subsequent computations. The time for a packet of minimum length is then 1029 μsec.

The poll of each secondary requires an exchange of minimum-length packets between it and the hub. The secondary must also wait at least 40 μsec before replying, to make sure that the hub has turned its receiver back on. Thus the time for a poll is 2106 μsec per secondary. Because of piggybacking, the first packet of data transmitted in each direction carries no additional overhead; only the I-field expands, and it adds 31.25 μsec per byte. With these observations, we can compute throughput under various assumptions.

Let the number of users on the polling list be S. The throughput is the number of information bits transferred per polling cycle, divided by the time to complete the cycle and further divided by 2, since two transfers are needed per packet delivered. If there are K packets transferred per cycle (all assumed to be piggybacked) and the average size of the I fields in these packets is N bytes, the channel throughput in kbit/s is given by the following formula:

$$\frac{256}{2} \cdot \frac{N \cdot K}{N \cdot K + 67.1 \cdot S}$$

where 67.1 is the polling overhead per secondary expressed in byte times. Some resulting throughputs for each of the K/2 multiplexed transfers are shown in Table 2 (in kbit/s). In entries of the form X/Y, X is the aggregate throughput and Y is the number of unidirectional packet streams.

The first line represents the best case for bulk transactions such as file transfers. The only offered traffic involves a single pair of secondaries, one sending and one receiving maximum length packets. The remaining lines describe what is likely to be more typical operation, with an average packet length of 100 bytes.

		S					
N	K	2	4	8	16	32	64
1500	2	122.5	117.5	108.6	94.3	74.6	52.6
1500	8	—	125.2/4	122.5/4	117.5/4	108.6/4	94.2/4
100	2	76.6	54.6	34.7	20.1	10.9	5.7
100	8	—	95.8/4	76.6/4	54.6/4	34.7/4	20.1/4
100	16	—	—	95.8/8	76.6/8	54.6/8	34.7/8

Table 2: Throughput (kbit/s)

	S					
K	2	4	8	16	32	64
0	15	25	46	88	172	340
8	—	—	108	150	234	402
16	—	—	—	213	297	465

Table 3: Latency (msec)

A reasonable measure of latency is the round trip time between two secondaries sending small packets (N = 32) in each direction. This determines how quickly the two can collaborate on demand-response style computations. The best case time is 2 polling cycles plus the final transfer time; the worst case is 3 cycles. Times for 2.5 cycles are shown in Table 3 (in msec). In this tabulation, K is the additional number of 100 byte packets transferred in each cycle.

These numbers and similar calculations show that aggregate throughput actually increases somewhat as the offered load increases, but the fraction available to each user drops. The average throughput per secondary cannot exceed 256/S kbit/s, but with large packets the unidirectional burst rate can approach 128 kbit/s.

Since our link-level protocol does not limit the amount of data exchanged per poll, there is no guaranteed upper bound on latency. As the figures above show, adding users drives up the latency even if they are offering no traffic. Most current applications have modest latency requirements, but low latency is critical to some proposed new ones. Keeping throughput per secondary high and latency low is the reason we hope that clusters will remain small and that overloading will be avoided by forming new ones.

C. Addressing and Routing

Two different kinds of addresses appear in each packet. The addresses SUA and DUA uniquely identify the originating station and the final destination. For our purposes, any identifier that is unique over a global address space consisting of the entire reachable network will do. Since call signs are required in any case, we propose to use them. Other alternatives such as IP addresses or Ethernet's 48-bit unique IDs would be equally suitable. Direct support of TCP/IP by using IP addresses and header formats would be very appealing. Since the performance of the polling protocol is very sensitive to minimum packet size, we have chosen instead to follow the Ethernet model, by requiring encapsulation of IP packets and eliminating the additional overhead at the link level.

Each packet also contains cluster-relative short addresses SA and DA. These cluster addresses are meaningful only within a single cluster (or collection of interconnected clusters administered as a single entity). They form a local address space and are attached to each packet as it enters the cluster. A possible source address (SA) is

- the unique cluster address assigned to an active secondary when it joins the cluster as described in Section D,
- an intercluster link, which is logically another kind of secondary and has a unique local address assigned or reserved by the hub,
- one (or perhaps several) control processes within the hub (for administrative and control packets),
- a null source (used by stations attempting to join the cluster).

A possible destination is any of the possible sources, plus reserved values for

- a broadcast destination ($ff, which is passed by all SCCs),
- a routing agent in the form of a process that runs within the hub.

In traffic sent by the hub (Direction = 0), DA is the selected secondary. SA identifies the secondary that originated the packet; it identifies the hub administrative port (poller) only if there is no traffic to forward. In a packet sent by a secondary (Direction = 1), SA is the cluster address assigned by the hub when the secondary joined the cluster. DA is the cluster address of the destination (local traffic) or of the agent managing the link to the destination (intercluster traffic) when such address is known to the source. Otherwise, it is the reserved address of the router port in the hub.

The cluster-relative address DA is important for two reasons. First, since the hub is broadcasting all traffic, the ability to do hardware-level filtering on DA takes a substantial load off the digital hardware in the secondaries. Secondly, DAs allow simple, fast intracluster switching by the hub. This is always desirable, and it will be essential in the higher performance switches we plan for the future. Since a DA is only 8 bits, forwarding decisions can be made by simple table lookup.

There are a number of ways that a secondary might supply the destination cluster addresses. One possibility is to defer to an intracluster router. To initiate an exchange with a station known only by its DUA, the unknown DA is replaced by a reserved DA that forwards the packet to a router. For intracell traffic, the router does simple lookup. If that fails, the router must choose a cluster address designating an intercell link or gateway (by a policy and mechanism not specified here). It then sets DA to that address and resends the packet. We expect the router to run on the hub hardware or on a computer tightly coupled to it; thus use of the router may add latency but does not generate extra RF traffic.

The SA field of the packet caters to another useful, low cost way of distributing cluster addresses: backward learning [10, pp. 297-298] by the secondaries. When a packet arrives, the secondary extracts the pair (SA, SUA) and stores that pair in some kind of associative memory (e.g., a hash table). Higher level software only understands UA addresses or their equivalents. For each transmitted packet, a low level of the packet driver does the lookup on DUA. If it succeeds (note that it always will after the first packet establishing a conversation), it inserts the correct DA; otherwise, it inserts the router DA.

Because of dynamic configuration, discussed in the next section, cluster addresses can be reused over time, and the (SA, SUA) pairs become stale. To deal with this, each pair is time-stamped when entered or re-entered; pairs with time stamps too old are periodically purged.

In this scheme, packets for the hub are normally not addressed to it, so its SCC must operate with address filtering disabled. Another consequence is this — a secondary that can hear another directly will get packets from it twice. The simplest way of dealing with this is for secondaries to discard all packets with the Direction bit set. We have some ideas for exploiting any direct paths that happen to exist, but they will not be part of our initial experiments.

D. Dynamic Configuration

The polling list is constructed and maintained dynamically. Unless the list has already reached its maximum permitted size, the hub periodically pauses in its polling of active stations and invites new stations to join the cluster. It does this by broadcasting a distinguished packet. Any station receiving such a packet can respond with another distinguished packet type that includes its

SUA. Multiple stations can respond. If the hub detects a collision, i.e., a carrier that cannot be demodulated or a packet with CRC error, it broadcasts that fact. Each station that responded is free to respond again if it first waits a random amount of time and defers to any other station that responds or is acknowledged first. Note that this is CSMA with a very long deaf period, i.e., close to Aloha. We expect cluster membership to change slowly relative to the rate of issuing invitations so that 0 or 1 response will be the norm. If this is not the case, we will use a protocol that converges to 1 response by successively narrowing the range of allowed SUA values.

When a hub hears a successful response to its solicitation, it allocates a cluster address for the new secondary, broadcasts an acknowledgment containing the SUA and SA, and adds the secondary to its polling list. Once a secondary hears such an acknowledgment of another station, it may not respond again until the initial invitation is repeated. If the selected secondary fails to hear the acknowledgment, it will continue to contend; the hub will either assign the same SA the next time it is heard or drop the entry by time-out.

Secondaries leave a cluster either by not responding to several successive polls, or by responding to a poll with an appropriate administrative packet.

Most of the states involved with joining or leaving a cluster have associated time-outs. Our experience indicates these time-outs must be carefully chosen [9]; we defer specifying the details pending some experimentation.

5. Where Are We?

Like many other projects done with spare-time labor and a very limited budget, progress has been erratic and is sometimes slower than we had wished. Our initial goal is to construct and operate 3 clusters in the San Francisco Bay area, one in Santa Rosa and the other two in the San Jose-Fremont-Palo Alto triangle. A prototype of the radio has been constructed and its performance has been demonstrated. We are currently trying to produce PC boards for the pilot run. The digital MIO board is a third generation descendent of the AIO design. At the time of writing, a wire-wrapped version is operational, PCB design is finished, and an initial batch of 3 boards is in fabrication. We also hope to borrow some hardware that will allow us to connect our network to an Ethernet and to experiment with digitized voice.

6. Acknowledgments

Conversations over the past several years with many amateurs helped to shape our ideas. Mike Chepponis K3MC designed the Awesome I/O board upon which the design of MIO is based and helped with the latter. Stu Phillips N6TTO provided much valuable advice on protocols and is developing some of the software.

7. References

[1] Chepponis, M., Elmore, G., Garbee, B., Karn, P. and Rowett, K., "The Implications of High-Speed RF Networking," *8th ARRL Computer Networking Conference Proceedings*, 19-29 [1989].

[2] Chepponis, M., and Mans, B., "A Totally Awesome High-Speed Packet Radio I/O Interface for the IBM PC XT/AT/386 and Macintosh II Computers," *7th ARRL Computer Networking Conference Proceedings*, 36-40 [1988].

[3] Elmore, G. and Rowett, K., "Implementation of a 1 Mbps Packet Data Link Using 10 GHz RF," *8th ARRL Computer Networking Conference Proceedings*, 38-42 [1989].

[4] Elmore, G. and Rowett, K., "Inexpensive Multi-Megabaud Microwave Data Link," *Ham Radio*, December 1989, 9-29 [1989].

[5] Elmore, G., "Physical Layer Considerations in Building a High Speed Amateur Radio Network," *9th ARRL Computer Networking Conference Proceedings* [1990].

[6] Elmore, G., "Prototype 905-MHz Digital Radio Completed," *QEX*, March 1990, 16 [1990].

[7] Reisert, J. H., "How to Design Yagi Antennas," *Ham Radio*, August 1987, 22-31 [1989].

[8] Saltzer, J. H., Reed, D. P., and Clark, D. D., "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, November 1984, 277-288 [1984].

[9] Schroeder, M. D., Birrell, A. D., Burrows, M., Murray, H., Needham, R. M., Rodeheffer, T. L., Satterthwaite, E. H., and Thacker, C. P., *Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-Point Links*, Research Report 59, DEC Systems Research Center, Palo Alto, CA [1990].

[10] Tanenbaum, A. S., *Computer Networks (2nd Edition)*, Prentice-Hall [1988].

Adaptation of the KA9Q TCP/IP Package for Standalone Packet Switch Operation

Bdale Garbee, N3EUA
Don Lemley, N4PCR
Milt Heath

Several new hardware systems intended for, or adaptable to, standalone packet switch use have appeared on the market in 1990. These include the Grace PackeTen, the Kantronics Data Engine, and the soon to be available AEA PS-186. One obvious use for these systems is implementation of a TCP/IP-based amateur packet network. This paper discusses some of the design issues uncovered in porting the KA9Q TCP/IP package to a standalone hardware environment, and will touch on some details of the implementations available now (or soon) for each of the mentioned hardware systems.

Background

For several years, Phil Karn KA9Q has spear-headed an effort to develop software based on the TCP/IP internetworking protocol suite for use on amateur packet radio. The "NET" package has enjoyed increasing popularity in countries worldwide.

As the number of users has grown, it has become increasingly obvious that certain limitations in the base software need to be addressed in order to permit the development of networks. Paramount among these are the use of static routing in most NET installations, and the dependence on end-user nodes to serve a dual function as network routers or gateways.

In some areas, the separation of end user and network gateway (or router) functions has been accomplished by reliance on existing TNC-2 based NET/ROM installations to provide network routing and "reliable" network service. This approach has helped to point out deficiencies in the routing algorithms and implementation used in NET/ROM, and has resulted in very poor performance in the areas where NET/ROM switches are configured in a single-port configuration, against all recommendation. In other areas, existing hardware platforms such as IBM PC clones and Atari ST systems have been pressed into service running the existing, standard NET software. This can be a very functional configuration from the standpoints of hardware availability and cost, software availability, and familiarity to network administrators. It can also be a real hassle, for reasons ranging from the lack of useable network routing and remote system management capabilities, to the seemingly mundane but very annoying issues of temperature range and power availability at switch sites.

The advent of more powerful digital processing hardware targetted for the amateur packet radio environment clearly is an attempt to address some of the problems related to use of existing user hardware platforms. Unfortunately, the new hardware addresses less than half the problem. The KA9Q TCP/IP software must be ported to these platforms, and some of the original software's design limitations, which are quite reasonable in an end user's system but are unworkable for a standalone diskless mountaintop switch, must be addressed and resolved.

Understanding the network model envisioned by the authors may help explain our motivation in developing NOS-based packet switches. In Figure 1, we show the typical "cloud-model" of a network. Attached to this network, and providing access service to several local user bases, are several switches. Each switch provides connectivity for zero or more AX.25 "terminal nodes" (as in TNC, or Terminal Node Controller), zero or more TCP/IP-based service providers (larger DOS or Unix machines), and zero or more TCP/IP-based user machines.

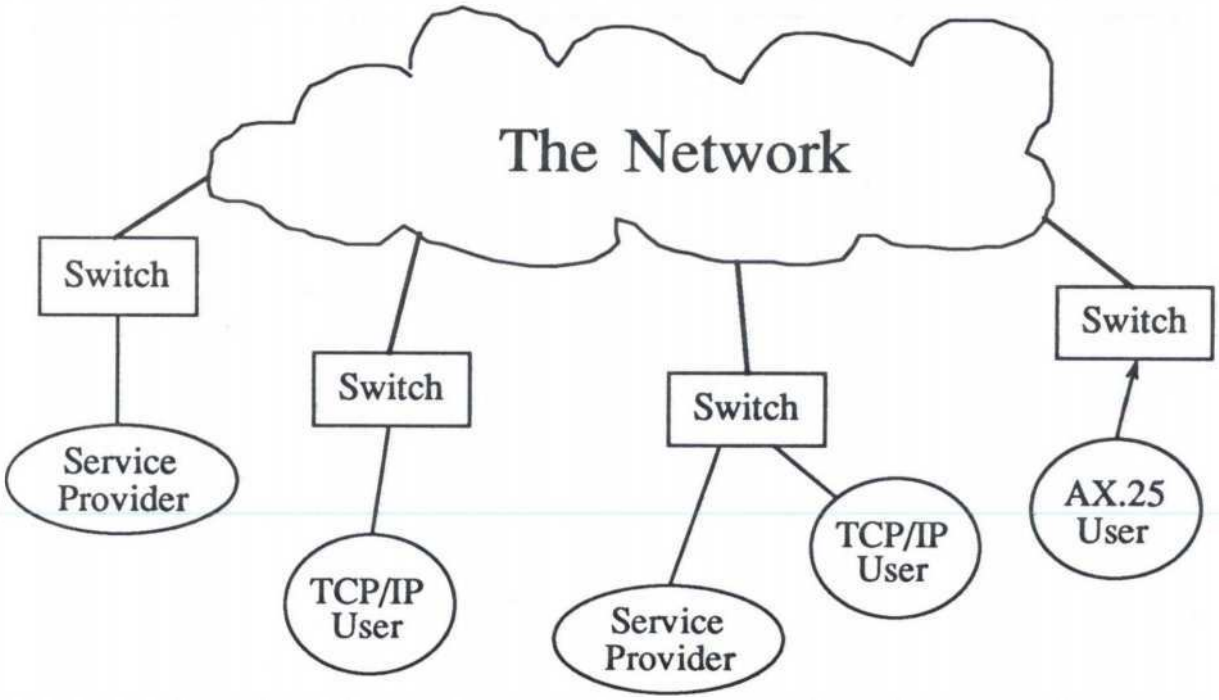


Figure 1: Our Model of an Amateur Packet Network

The most important points about this diagram are that individual users and their machines need know nothing at all about the topology of "the network", and that a switch can provide service to any number of AX.25 or TCP/IP users.

For the TCP/IP users, operation is exactly as might be expected. Sessions can be initiated to any number of servers, other systems can initiate sessions to local servers, etc. Adding a new user's host at maximum requires knowledge of the local switch address, and even that may not be required if user nodes run automated routing protocols too. Connections are established through the network by knowing only the address or symbolic name of a remote system, and the service desired on that system, which could be further simplified by the addition of a location broker on the network.

For an AX.25 user, the local switch appears to be the service provider. A user connects to the local switch with a normal AX.25 connection, and is presented with a menu of available services. Upon selecting a service, a TCP-based connection is automatically opened from the local switch, across the network, to a remote service provider. Services could include the ability to login to a BBS or Unix machine, where a full range of options exist. Or, services could be as directed as obtaining the current weather report from the National Weather Service, looking up a callsign in an electronic callsign server database, etc. The important point is that AX.25 users are fully supported as network citizens through the terminal-serving capability of their local switch.

The implementation of KA9Q's NOS TCP/IP software on the platforms discussed in this paper has become commonly known as "NOSINABOX", and will be referred to by that term for the remainder of the paper.

Issues

The initial implementation of NOSINABOX has focused on the fundamental changes required in the KA9Q TCP/IP package to allow it to function in an environment without disk drives, operating systems, and direct human intervention. This has involved disabling services that make no sense in a standalone environment, rewriting facilities that are necessary regardless of environment but which depend on resources unavailable to us, and adding new capabilities that are uniquely necessary in a standalone, remote hardware environment.

The NOS software package includes protocol support

for a variety of functions that only make sense in the presence of users, file systems, or both. These include the FTP file transfer protocol client and server, the SMTP electronic mail client and server, and the Finger server among others. Phil provides a compile-time mechanism to eliminate all servers from the package, but additional effort was required to locate and disable code for nonsensical clients.

One of the features of NOS versions of the KA9Q software is the ability to query domain name servers such as BIND for hostname to IP address translation. Unfortunately, the stock name server client provided in NOS assumes the availability of a file system for caching the results of previous name queries. This code required a major rewrite, and included modifications made by PA0GRI and others to the stock NOS implementation.

File system references also appeared in the initial configuration section of the software. In a PC or workstation environment, NOS loads configuration information from a disk file that the user provides. Considerable effort was expended to provide fundamental configuration from ROM in all versions of NOSINABOX. Because we take advantage of special hardware features available on the different systems, this is discussed further in a later section.

The standard NOS software was written assuming that a keyboard and display local to the processor would be used as a "console" for the package, providing a configuration and debugging interface to the software. This is not true in most packet switch environments, where the switch lives on top of a mountain reachable only on snow-shoes much of the year, or high on a tower requiring professional assistance to access. Fortunately, as part of the NOS rewrite, Phil based the console interface to the software on the same "sockets" mechanism used to attach applications to networking services. It was therefore relatively easy to provide a mechanism for remotely attaching to the console over the network. Rudimentary but long-term inadequate security for this facility is provided in the initial release.

Implementation-Specific Issues

Since the author's efforts in developing standalone versions of NOS have been carried out independently due to differences in the hardware platforms involved, and geographic location, the actual feature sets provided by the implemented code are somewhat different. However, considerable effort has been expended to ensure that at least philosophically the versions of NOSINABOX are aligned in such a way as to provide compatability for future development of new features. The following paragraphs outline some of the software features of each NOSINABOX version as of this writing.

Grace PackeTen

Because the Grace Communications PackeTen is based on the Motorola 68302 "Integrated Multi-Protocol Processor" instead of the Intel platform traditionally supported by KA9Q's NOS, much effort was expended porting the code to the new processor. This effort involved both making a standalone version of the code function, and optimizing the software to take advantage of powerful new 68302 features. The resulting version of NOSINABOX has been dubbed "NOS302".

NOS302 is the NOSINABOX version most fully developed at this time. In addition to the feature set neccessary for minimal standalone operation, the NOS302 system provides:

1. Software Watchdog Function

   This feature implements a software sanity check within the NOS package. Since a remote packet switch is by definition not easily accessable, it is imperative that some form of dynamic software integrity test be performed during normal operation. The purpose of this test is to prevent a system which has suffered a software or firmware failure from "going off in the weeds" and becoming nonfunctional forever, requiring some poor soul to reach the site and manually restart the system. This "NOS independent" software "watches" the system software, and if for some reason NOS is not running through it's normal routine, the system is restarted with a hard reset.

2. EEprom Configuration

   Since amateur packet switches are usually inaccessable, they must be capable of surviving a power glitch or outage, rebooting to a known functional state appropriate for the site. Since the PackeTen switch hardware provides non-volatile memory in the form of EEPROM, menu driven software is provided for configuration of such options as IP address, AX25 address, link definitions, and other neccessary site specific information.

3. Attachable / Detachable Console Port

   In order to avoid wasting one of the serial links for use as a local console during operation as a standalone switch, a "Console" device was devel-

oped which can be attached and detached just like all other links. This allows a remote switch to use all links during normal operation, but if on-site service of a node is required, a simple jumper strap will configure one of the ports as a local console for debugging.

4. Scatter Gather Transmit

   This feature involves the use of an advanced 68302 capability to dynamically build a transmit frame "on the fly" from a list of buffer pointers. The major advantage of this capability is that the data does not have to first be copied from NOS's "mbuf" chains into a contiguous data frame area before transmission, thereby decreasing processor overhead, and increasing throughput on a link. It is Worth noting that KA9Q's approach to the problem of layered protocol frame construction, namely mbuf chains, fits perfectly with the design of the 68302 serial communications controllers. His design allowed the "scatter gather" feature to be implemented relatively easily, and efficiently.

5. Receive Mode Buffer Chaining

   Receive buffer chaining is another very important feature of the 68302 platform. The 302 serial channels are programmed with a list of available receive buffers. Then they begin to accept incoming frames on the link. Upon completion of a received frame, they simply mark the frame as ready for processing, and then proceed, utilizing the next available receive buffer. All of this functionality is performed with no main processor intervention, or interrupt service required. Since the serial channel's buffer chain list is up to eight (8) buffers deep, the main processor is only required to service a receive interrupt every 8 frames in order to prevent receive data overrun on a link.

6. Software Timer Services Added to NOS.

   It was considered important in the 68302 NOS implementation to be able to handle "multiple" high speed data links, without the use of wasteful polling loops for such things as keyup/keydown radio control. Therefore software programmable timer services were added to NOS to allow the programmer flexibility in writing device driver timing routines.

7. Automatic Power Conservation Mode

   One of the features of the MC68302 processor is the ability to cut it's power consumption from a typical 60 ma down as low as 1 ma in it's lowest power mode. The current NOS302 release 1.2 implementation supports this type of operation, allowing consumption to be reduced by approximately half, to around 30 ma, during idle periods between received packets.

Kantronics DE and AEA PS-186

NOSINABOX for the Kantronics Data Engine and AEA PS-186 is under development at the time of this writing. Because the processor used (V40 and 80C186, respectively) and I/O hardware configuration (V40 DMA to 8530, and NEC DMA to 8530, respectively) are so similar, the two platforms are being treated as functionally equivalent, differing only in the number of ports (2 or 4) that are provided in addition to the dedicated console port.

Despite the inclusion of battery-backed CMOS RAM capability on both of these systems, the initial release of NOSINABOX will probably include a configuration utility intended to customize the ROM images to reflect configuration information for a given site. Configuration changes will of course be possible after boot from the system consoles.

The initial release also will not take advantage of the DMA capabilities of these systems. Because the DMA controllers used do not provide a "scatter gather" chaining mode identical to the 68302 version, additional effort will be required to interface the NOS mbuf handling to DMA buffers for each port. For this reason, initial performance of these two platforms in interrupt-driven operation will be far from spectacular.

Both the AEA and Kantronics platforms include hardware watchdog functionality, and NOSINABOX will use this functionality to force a hardware reset after software crashes.

Future Directions

Once fundamental hardware-specific issues are resolved, the authors hope to integrate the two currently divergent implementations of NOS for standalone hardware into a single NOSINABOX environment for multiple hardware platforms. It is highly likely that the Grace PackeTen system will always be "one step ahead", since it is a commercial product with TCP/IP protocol support provided by the manufacturer. The TCP/IP support for the AEA and Kantronics products is currently available due to entirely volunteer effort.

A growing concern in the commercial and educational network markets that is or will soon be mirrored in the amateur packet environment is the issue of network monitoring and management. The SNMP Simple Network Management Protocol has recently come into wide use as the mechanism of choice for interrogating and controlling network entities. Several vendors are now producing slick, graphically-oriented user interfaces for network management that rely on the SNMP protocol for communication with diverse networking hardware systems from multiple manufacturers. The effort required to implement at least a subset of SNMP target functionality seems quite acceptable, and it is highly likely that we will add SNMP support in a future release of NOSINABOX.

Considerable room for experimentation exists in the area of dynamic route determination. Fred Goldstein, K1IO, has drafted a specification for a protocol called RSPF, or Radio Shortest Path First. A preliminary implementation for NOS was written some months ago by Anders Klemets, SM0RGV, and we are currently investigating including the RSPF code in the next NOSINABOX release. RSPF has several features that make it an attractive potential alternative to the routing provided by NET/ROM and the RIP protocol currently included in NOS. One of the keys to automatic, dynamic route determination on radio links is the issue of determining link quality, and this is an area that will no doubt be addressed at great length over time.

Currently, a very simplistic approach is used to "secure" the remote system administration facility. An alternative that intrigues the authors is the adoption of an IP-level option to cypher-checksum each frame sent on a secure connection. This approach involves encrypting each packet as it is assembled for transmission, calculating the checksum of the encrypted packet, and transmitting it along with the plaintext packet and normal data corruption checksum. By performing the same process on incoming packets, a receiver could know positively whether the sender knew a pre-negotiated key, thereby validating control packets as coming from an authorized control station. This is a clearly legal use of encryption for authentification purposes accepted by the FCC, with no restrictions since the actual data content of each packet is sent in cleartext. An implementation of this option, and a requirement for use of the option for remote sysop functions, would be a very secure way to control access to the configuration of a switch site.

Each of the hardware platforms discussed in this paper include some mechanism for the inclusion of custom, "expansion" hardware. One possible use of this facility would be to drive hardware interfaces for telemetry and control. It might be much easier to obtain permission to co-locate a switch with an existing voice repeater facility if added value could be offered in the form of a secure remote control and monitoring station for repeater equipment, power equipment, or weather data collection hardware, as examples. This area of functionality will be investigated for future releases of NOSINABOX, either as support for specific future hardware add-ons to the existing platforms, or perhaps as a generic interface for driving custom hardware at the I/O address and data level.

The exact performance of NOSINABOX on all of these hardware platforms is not yet known. However, the PackeTen has as of this writing been placed in service running 56 kb/s full-duplex radio links as well as having been tested with 2 megabit/s full-duplex wire-line links. In one configuration it has simultaneously supported six (6) 56 kb/s links, two (2) links at 9600 and two (2) others at 1200 baud. As more NOSINABOX implementations become available and are placed into service, the additional performance information obtained will be used to further redesign and optimize the internal data handling algorithms, thus providing additional performance improvements.

We do not currently fully understand all of the relationships of hardware architecture, processor family, and protocol design on the real aggregate throughput of NOSINABOX packet switches. Gaining and documenting additional knowledge in this area will help potential customers make intelligent decisions about which switch to buy for a given environment, in addition to fueling continued performance enhancements.

Different approaches to the "terminal server" functionality provided by NOSINABOX have been proposed, including a suggestion by WB6RQN that AX.25 users be treated to an automatic, blind "rlogin" connection a local Unix server when a connection to the local switch is established. These ideas deserve further investigation, and it is likely that a future release of NOSINABOX will provide support for multiple approaches to solving the continuing problem of AX.25 support in a computer-to-computer networking environment, with some mechanism provided to configure the mechanism desired for a given local area.

Conclusion

This paper has discussed issues related to use of the KA9Q "NOS" TCP/IP software in a standalone, amateur packet radio packet switch environment. Current status and future directions of the authors' efforts in this area have been reported.

For continuing progress reports and other information about the effort to develop intelligent packet switch systems, the reader is directed to "Packet Status Register", the newsletter of TAPR, the Tucson Amateur Packet Radio organization. The authors may be reached by a variety of means, listed below.

We strongly encourage you to provide feedback on the ideas expressed in this paper, and encourage you to acquire and experiment with one or more of the packet switch platforms discussed, and report your experiences to us

Contact Information

Bdale Garbee, N3EUA
4390 Darr Circle
Colorado Springs, CO 80908
bdale@col.hp.com
CIS: 76430,3323

Don Lemley
623 Palace Street
Aurora, IL 60506
donl%ldhmi.uucp@col.hp.com
CIS 73230,310

Milt Heath
75 South Chestnut Street
Aurora, IL 60506
milt%ldhmi.uucp@col.hp.com

NETWORK ROUTING TECHNIQUES AND THEIR RELEVANCE TO PACKET RADIO NETWORKS

James Geier Martin DeSimio, WB8MPF Byron Welsh, KD8WG

Air Force Institute of Technology
Department of Electrical and Computer Engineering
Wright–Patterson AFB, OH 45433

Abstract

With packet radio networks, the distance between source and destination nodes typically necessitates one or more nodes to relay data to the final destination. Thus, some form of routing must take place. This paper explains several current network routing algorithms and shows their relevance to packet radio networks. In addition, current research at AFIT concerning the development of an automatic routing algorithm for Air Force Logistics Command's (AFLC) HF packet radio network is explained.

1 INTRODUCTION

Most routing algorithms store node address information in tables, which show the next node to receive a packet. These routing algorithms may be static or dynamic. If the algorithms are static (nonadaptive) the table entries do not change during normal operation of the network. Dynamic (adaptive) routing algorithms periodically update the tables to reflect changes in the network's topology or utilization or both [6].

Since packet radio networks typically have changeable connectivity, the routing mechanism must be capable of updating routing tables. This paper examines three basic types of adaptive routing algorithms: centralized, distributed, and isolated [7]. The final section describes current research of an automatic routing algorithm for AFLC's HF packet radio network.

2 CENTRALIZED ROUTING

A centralized routing algorithm requires a routing control center (RCC) to make routing decisions based on information gained from each node within the network. Each node monitors connectivity and delay metrics among neighboring nodes and periodically sends this information to the RCC. The RCC calculates the best route (normally in terms of least delay) and sends each node new routing table information depending on the most recently measured state of the network.

In most cases, a source node needing to send data packets can notify the RCC of the source and destination. The RCC will respond with a special call request packet called a needle packet, that contains the route which is the most efficient circuit. The route is specified as an ordered set of nodes. The source node then sends the needle packet through the network to establish the circuit, and then data packets can follow.

Although centralized routing offers a solution to adaptive routing, this technique has disadvantages worth discussing. For one, centralized routing requires a large amount of overhead due to routing information sent between nodes and the RCC. As a result, centralized routing may not be suitable for some networks operating with limited bandwidth, such as HF packet radio. Also, large networks having many nodes will require the RCC to perform lengthy calculations to determine optimum routes. Hence, the "optimum" table entries may not be valid if the network topology changes rapidly.

3 DISTRIBUTED ROUTING

With distributed routing, each node distributes routing metrics (connectivity information, node delays) throughout the network, enabling other nodes to update routing tables. Distributed routing has proven to be very robust with ARPANET (Advanced Research Project Agency Network). Within ARPANET, each node periodically measures the delay to each node within one transmission hop and puts this information into a status packet. Nodes within one transmission hop are known as neighbors. The node then transmits the status packet to each neighbor, which records the status information. Each neighbor repeats the delay measuring process, formulates a status packet containing local delay information as well as delay information from incoming status packets and sends this status packet to each of its neighbors. By having all nodes follow this process, each node will eventually have an overall "picture" of the network in terms of node-to- node delays. Each of the nodes can then determine the route of least delay by referring to the status information received from other nodes.

Distributed routing requires a significant amount of overhead with packet radio networks having highly mobile nodes. Each node must send status information often enough to account for rapid changes in topology due to node movement. Hence, the network must have a considerable amount of bandwidth available or suffer from rather low throughput.

Jubin and Tornow explain that the DARPA (Defense Advanced Research Project Agency) packet radio network (not ARPANET) applies distributed routing techniques by having each node maintain a tier table [2]. The tier table specifies nodes that are one hop away (tier 1), two hops away (tier 2), three hops away (tier 3), and so on. The tier table is arranged in a matrix format. The tiers represent the rows, whereas the tier 1 entries head off the columns. For example, node X could have a tier table as shown in Figure 1. Here, nodes A, B, and C are neighbors of node X; nodes D and E are neighbors of node A; node N and J are neighbors of node D; node F is a neighbor of node B, and

so on. If node X has a packet needing transmission to node F, node X would choose to send the packet to node F via node B because the transmission would take only two hops in comparison to three hops if sent via node C. The tier table gives information regarding connectivity among the nodes; therefore, software at the source node may use the tables to make effective routing decisions.

The nodes update their tier tables in the following manner. Every 7.5 seconds, each node transmits a Packet Radio Organization Packet (PROP) that announces the node's existence and includes a copy of its tier table. A node receiving a PROP simply includes the information in its own tier table. After a period of time, all nodes in the network will have complete connectivity information for the entire network.

Very high frequency (VHF) packet networks established by amateur radio operators also employ distributed routing techniques. Amateurs use an approach similar to the DARPA packet radio network described above. Each node in the network periodically broadcasts information concerning the nodes it has connectivity to. Included in the broadcasts is not only node topology information, but also a relative measure of route quality within the network. The route quality between two nodes is calculated based on the type of link between the nodes in question. Factors that affect node quality include the baud rate supported by a link as well as whether it is a radio or a direct hardware link. Overall link quality is calculated by multiplying the qualities of each link traversed in the path from the source to the destination. A node will choose the route having the highest quality which insures that the route used will support the highest baud rate possible as well as the route with the least number of hops. Nodes are also automatically deleted and added to the network via the same broadcasts [5]

4 ISOLATED ROUTING

Isolated routing allows a node to make routing decisions without reference to an RCC or other nodes. Three classical approaches of isolated routing are flooding, hot potato and backward learning. [7]

4.1 Flooding

With the flooding technique, a node with a packet to send initializes a hop counter in the packet header to a count of zero and transmits the packet out every outgoing link. With packet radio networks, this means that the node simply broadcasts the packet. Each node within the network examines incoming packets, and if the packet has not reached its destination and the hop counter has not reached a predetermined limit, the node increments the counter and rebroadcasts the packet. The hop counter maintains the stability of the flooding routing algorithm. Without the hop counter, nodes would continue to retransmit packets indefinitely. Since every node follows the flooding process, the packet will eventually reach its destination.

Flooding is effective as a routing technique with networks having rapidly changing connectivity. This reasoning stems from the fact that flooding does not require routing updates as the network changes. Even though flooding has overhead in the form of redundant data packet transmissions, the amount of overhead saved due to the absence of routing table updates outweighs the amount of overhead flooding generates in retransmissions for systems with dynamic topologies. However, with static topologies the opposite condition occurs; the amount of overhead due to redundant transmissions is greater than routing table updates because static topologies do not require as many routing table updates.

Flooding, because of its nature, offers the shortest end-to-end packet delay when compared to other routing techniques. Since control packets normally require expedient delivery, even packet radio networks having static topologies use flooding to disseminate urgent control information, regardless of the fact that it will generate higher overhead.

4.2 Hot Potato

The hot potato routing method bases routing decisions on the current availability of transmit queues within the node. Pure hot potato will put an incoming packet in the outgoing queue having the least number of packets waiting for transmission. Thus, the current node treats the packet as a "hot potato" by getting rid of it as soon as possible, without regard to where the packet goes next. Eventually, the packet will reach its destination.

Static routing, which uses routing table entries that do not change, can effectively utilize hot potato by putting a packet in the second best (or lesser) choice output queue if the best choice output queue has more than a certain number of packets waiting for transmission. As a result, congestion will be kept to a minimum on the best choice lines.

4.3 Backward Learning

The backward learning approach assumes data packet headers contain source node addresses and hop counters. The source node initializes the hop counter to zero before transmitting the packet, and intermediate nodes increment the hop counter by one before retransmitting the packet. For the purpose of updating routing tables, each node constantly monitors the incoming packets by noting the packet's original source address, hop count, and address of the immediately preceding node. With this information, a node can make educated decisions on which node to send outgoing packets to for delivery to a specific location. A node transmits a packet to the neighbor where packets came from with the least hop count and originated from the desired destination node.

The main advantage of the backward learning technique is less overhead because the data packets themselves carry the routing information; bandwidth-consuming routing control packets do not need transmission. Of course, the backward learning algorithm assumes network conditions are approximately the same in both directions. Also, the routing tables will only be accurate if data are sent between the nodes frequently because the data

packets are responsible for carrying the routing information.

The choice of routing algorithm for a packet radio network depends greatly on the mobility of the nodes and the available system bandwidth. If the network incorporates highly mobile nodes, the flooding technique is desirable because the changing topology, when compared to a static one, will not cause the routing algorithm to generate additional overhead. If the network is somewhat stationary, then a distributed routing method works effectively because periodic status transmissions can enable the updating of tables as node availability changes due to changes in radio wave propagation. On the other hand, a network with highly mobile nodes and limited bandwidth may benefit from a backward learning technique if there are frequent data transmissions between nodes. Whichever routing algorithm is chosen, it should be the most efficient in terms of least delay.

5 CURRENT RESEARCH AT THE AIR FORCE INSTITUTE OF TECHNOLOGY [1]

The Air Force Institute of Technology (AFIT) is continuing in the development of an HF packet radio network for Air Force Logistics Command (AFLC) for the transmission of text messages during wartime contingencies and natural disasters. This network will eventually consist of a packet radio station (node) at each of AFLC's Air Logistics Centers and several portable units. AFLC expects to use the packet radio system during wartime if other message systems fail because HF packet radio architecture, due to relatively low operating frequencies, lends itself well to post-nuclear attack. In addition, the ease of mobility with packet radio nodes makes the packet radio system ideal for supporting communications during the response to natural disasters [3,4].

Each node in AFLC's packet radio network consists of an Advanced Electronics Applications, Inc. PK-232 Multi-Mode Data Controller, which acts as a terminal node controller (TNC). In addition, each TNC connects to an AN/URC-119 (HF) broadcast radio that prepares the data for transmission through the atmosphere. A software interface, written at AFIT, interfaces with the TNC. The interface allows an operator at a node location to send text messages to other node sites. The TNC controls the HF radio and accepts commands from the operator via the software interface [4].

The current network software insists that the operator at a node location in AFLC's packet radio network must specify other nodes that comprise a circuit between the source and destination before sending a message. The source TNC then connects to the destination via the chosen nodes, and the nodes along the chosen path relay the message to the destination. In order to make wise decisions on the selection of routing nodes, the operator must manually, by phone calls to node sites or trial-and-error transmissions, determine the availability of nodes. System operators can keep a log of previous successful routes and store these in existing routing tables; however, the availability of nodes fluctuate as conditions for radio propagation change. Thus, routing tables may be inaccurate. Due

to the slow response time of the human operator in choosing relay nodes, this method of manual routing selection degrades network performance [3].

An AFIT research objective is to develop and implement a method that will free the system operator from selecting nodes that relay messages in-route to the destination and allow the system adaptive to changes in node availability. The proposed method will be in the form of a software algorithm that will automatically monitor the availability of nodes in the network and determine the best route that a particular message should follow to the final destination. Thus, the operator will only need to specify the destination node to successfully send data to another node.

The general class of routing chosen for the automatic routing algorithm has a distributed form similar to the algorithms used by the DARPA and VHF amateur radio packet networks. Centralized routing was not chosen because of the amount of overhead generated, and even though isolated routing does not generate very much overhead, it was not acceptable in this case because of the dependence of data transmissions to carry routing information.

5.1 Routing Table Structure

Each node will initially establish, and continually update, a tree-like routing table. The general form of the routing table is shown in Figure 2. The node at level 0 (resident node) identifies the node that contains the routing table, and one or more branches may form below this level. The level 1 entries are immediate neighboring nodes, which are one hop or radio link away from the resident node. The resident node will have as many level 1 neighbors as there are nodes having direct connectivity with the resident node. Neighbors that are two hops away from the resident node are identified as level 2 neighbors, which are shown directly connected to respective level 1 neighbors. Levels 3 and higher may also form depending on the size and connectivity of the network. For instance, a resident node will have a routing table entry for level 10 if it has connectivity with another node 10 hops away.

An example of the routing table organization is shown in Figure 3. This figure illustrates a simple connection of nodes A, B, C, D, and E and a depiction of the routing table for nodes A and C. Notice that the structures of the routing tables match the network's topology. In the routing table at node A, nodes B and C are shown as level one neighbors. The table shows that node A has direct connectivity with nodes B and C. The routing table at each node also represents the connections between node B and nodes C, D, and E, which are two hops away from node A. The same is true for the connections between node C and nodes B and D, which are also two hops away from node A. In addition, the table correctly denotes the links among nodes B and D with node E. Hence, the routing table gives a clear picture of the organization of connections among network nodes.

5.2 Route Determination

Using the routing table structure above, a node can choose the circuit that provides the connection that offers the least number of hops between the source and destination nodes. For example, if node A in Figure 3 wishes to send a packet to node E, then node A can send the packet three possible ways. It could use circuit A-C-B-E, A-C-D-E, or A-B-E. It is obvious that circuit A-B-E would be the best selection because this circuit will require only two hops versus three hops for the other choices.

A general rule for deciding which circuit to choose is as follows. A node needing to send a packet to a certain destination begins by scanning each level of the routing table starting with level 1 and proceeding to higher-numbered levels until the destination node is found. The resulting level number will indicate the shortest number of hops to the destination. To determine the route, the algorithm starts at the destination node found in the table and follows the path back to the resident node. The nodes that the algorithm must traverse to get back to the resident node comprise the desired circuit. If the algorithm finds more than one node on the same level that identify as the destination, then the algorithm will choose one at random and then determine the circuit, as explained above, by following the path back to the resident node.

If for some reason the network can not support a connection by way of the chosen circuit, then the algorithm will choose another circuit based on the same routing table level as before if another destination node entry resides at that level or drop down through the truth table to higher-numbered levels until another destination node entry is found. In addition, the algorithm will initiate a probing that will determine which part of the circuit is causing the original circuit to be faulty. The algorithm will accomplish the probing similar to the method technicians use to manually troubleshoot a faulty circuit. First, the resident node will fabricate and send a probe packet that is addressed to an intermediate node at or near the center of the faulty circuit. If this node is available and still connected to the circuit, it will send an immediate response to the sender. If the resident node receives a response from the intermediate node, the resident node can assume the circuit is operational up to the that point in the circuit, and if the resident node does not receive a response, the resident node can assume the problem lies somewhere between the resident and the intermediate node. The resident node will continue probing in the appropriate direction until a faulty node pair is found.

As an example of the probing action, refer to Figure 3. Node A will assume has found circuit A-B-D is faulty if no acknowledgment heard from node D. Node A will then send a test probe to the midway point, which is node B. If an acknowledgment is not heard within a specified time period, then the problem must lie between nodes A and B. With this example, two possible causes of the discontinuity are that either node B is inoperable or nodes A and B have lost direct connectivity with each other. Because of the sporadic propagation of HF radio signals and the reliable nature of nodal hardware, the most probable cause is that the two nodes have simply lost connectivity due to a change in

radio propagation. Thus, the algorithm at the resident node should delete from the routing table any A-B or B-A node pair and the connections that fall directly below the faulty node pairs. This will avoid the selection of other circuits that contain the faulty node pair. Once the proper deletions have been made, the algorithm will broadcast a status packet that identifies the changes made to the table.

5.3 Updating Routing Tables

Nodes establish and periodically update routing tables through the use of status packets. Status packets contain the sender's identification and a copy of the sender's routing table. Initially, a node broadcasts a status packet which its neighbors receive. Nodes do not retransmit status packets. When a node receives a status packet, the node determines which node sent the packet and then does one of two things. If the origin of the status packet is not currently a level 1 table entry, the (resident) node will append the origin's table to the existing table, with the origin as a level 1 entry. If the sending node is already a level 1 table entry, then the resident node will replace the existing table's entries (associated with the sender) with the sender's table included in the status packet. If the routing table update causes the table to change, then the node will broadcast a status packet.

An important function of the routing algorithm, with regard to table updates, is to determine when a level 1 neighbor has become disconnected from the resident node. If the neighbors of an inactive node are not warned, then they may attempt to send data packets through the "dead" node. The solution to this problem is to have each node periodically send status packets if no information packets have been sent over a certain period of time. Thus, the resident node can monitor nodes currently listed as level 1 neighbors, and if the resident node does not receive any activity from a certain neighbor in the specified period of time, then it will delete from its routing table that level 1 entry and all connecting branches and nodes listed below it. In addition, the resident node will broadcast a status packet to reflect the change of connectivity. This will reduce the chance of a node choosing a circuit that includes the "dead" node.

The amount of time to allow between periodic status transmissions is a function of the connectivity variance among nodes. Nodes should send status packets often enough to account for changes of connectivity due to variations in HF radio propagation. HF propagation is sporadic, but it tends to change over a period of hours, not minutes or seconds. Thus, the transmission of status packets for the determination of connectivity changes should occur at least once every hour if no information packets are being sent. Notice that this mechanism alone will not allow quick modification of the tables due to nodes that become inoperable.

With this method of updating the routing tables, a resident node is capable of gathering enough information to make routing decisions, but the tables will have a great deal of redundancy, mainly in the form of looping. Consider again the example illustrated in

Figure 3. When node C broadcasts its status packet, it effectively sends its routing table to nodes A, B, and D. In effect, node A will replace its level 1 entry corresponding to node C with the table contained in the status packet from node C. As a result, the table at node A will appear as in Figure 4(a). The redundant information here includes the new parts of the table that include node A, the resident node. For instance, node A will never need the circuit A-C-A-B-E, A-C-A-B-D or A-C-B-A because they will loop data packets through the sending node (node A) one time before reaching the destination. This looping introduces useless overhead in the network.

Routinely, a node inherently sends redundant looping information when sending a status packet. The reason for this is that the status packet contains a copy of the routing table of the sender which normally includes the receiving node's level 1 entries. This causes the looping redundancy in Figure 4(a). In fact, if nothing is done, continual status transmissions will not only cause looping in the tables, but it will make the routing tables grow in length without bound.

5.4 Pruning

A method to reduce the looping and endless routing table growth is to "prune" the table after using the status packet to update table entries. The objective in pruning is to delete any parts of the routing table that represent circuits with a loop through particular nodes (not only the resident node). By eliminating all circuits that cause this looping, the resulting pruned table will not grow indefinitely, and it will provide just enough redundancy to accommodate valid alternate routes.

The rules of pruning are straightforward. After receiving a status packet and incorporating the neighbor's routing table as part of the existing table, the resident node searches for repeating node entries in every circuit. If the resident node finds a repeating node, it deletes that node plus any branches that fall below and connected to the repeating node. For example, the resident node (node A) of Figure 4(a) will not find any repeating nodes in circuits A-B-E, A-B-C, A-B-D, A-C-B-D, A-C-B-E, A-C-D-B, or A-C-D-E. However, it will find repeating nodes in circuits A-C-A-B-E, A-C-A-B-D, and A-C-B-A. After deleting the repeating nodes and all connecting nodes below, the routing table at node A will appear as shown in Figure 4(b). This routing table illustrates the desired result: no looping redundancy.

After pruning, the resident node will immediately broadcast a status packet if the new routing table structure is different than before receiving the last status packet. This ensures other nodes get the updates as soon as possible.

5.5 Adaptability

In order to discuss the adaptability of this algorithm, it is important to first outline what causes connectivity to change and then treat each cause separately. Network connectivity

may vary if a node is added to the network, if a node is deleted from the network, or if the radio propagation between two nodes changes.

If a node is added to the network, a technician will initialize the node, and the algorithm will clear its routing table, enter the proper resident node identification in the table, and immediately broadcast a status packet. This status packet will only indicate the node sending the status because the rest of the table will be empty. After reception of the status packet, each of its neighbors will immediately broadcast status packets as well so the new node can learn the connectivity among other nodes and other higher level neighbors can learn about the new node. Eventually, the new node will know the connectivity of all other nodes in the network, and the other nodes will know about the new node. Therefore, the algorithm will support the addition of a node.

If a node becomes inoperable, its level 1 neighbors will discover this because the neighbors will have been listening for transmissions from the "dead" node. After a period of time, if a node has not received any information packets or periodic status transmissions from a specific level 1 neighbor, it will delete that neighbor from its routing table and send status packets to reflect the changes so other nodes can adjust their routing tables. Because a node may not transmit but still be operable, each node must periodically broadcast a beacon so all neighboring nodes will sense the presence of a possibly idle node. Thus, the algorithm will accommodate the removal of a node.

As radio propagation changes between a pair of nodes, the connectivity between that pair of nodes may be lost. This disconnection may possibly affect every node's routing table. If the loss of connectivity is between, say, nodes A and B, then node A will not hear anything from node B, and node B will not hear anything from node A. Therefore, after a period of time, node A will delete from its table node B, and node B will delete from its table node A. Of course, this will initiate the immediate transmission of status packets, which will reflect the changes. Thus, the algorithm will successfully update routing tables as connectivity changes.

6 CONCLUSION

Designing a routing algorithm for a packet radio network is challenging, mainly because of changing connectivity due to node mobility and variances in radio propagation. Hopefully, this paper will have given networkers a basis of knowledge concerning routing techniques as they relate to packet radio networks.

AFIT's routing algorithm is currently being written in the Turbo C programming language and will be incorporated into the existing user interface. To handle the transmission of status packets, the software will operate the TNC in a connected mode to use the TNC's error recovery mechanisms. AFIT will begin testing the revised user interface with AFLC's packet radio network in October 1990.

Bibliography

1. Geier, James T. "Automatic/Adaptive Routing Algorithm for the Air Force Logistics Command Packet Radio Network." MS thesis draft, AFIT/GE/ENG/90S. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, July 1990.

2. Jubin, J. and J.D. Tornow. "The DARPA Packet Radio Network Protocols," Proceedings of the IEEE, 75: 21-32 (January 1987).

3. Katsampes, TSgt, NCOIC Operations and Plans. Telephone interview. Air Force Logistics Command, SCSXP, Wright-Patterson AFB, OH, 13 March, 1990.

4. Marsh, Steven L. "Integration of the Air Force Logistics Command Packet Radio Network." MS Thesis, AFIT/GE/ENG/89D-29. School of Engineering, Air Force Institute of Technology (AU), Wright- Patterson AFB OH, December 1989.

5. *NET/ROM for the TNC-2.* Users' Manual. Software 2000, Inc. Arroyo Grande, CA, 1987.

6. Schwartz, M. *Telecommunication Networks.* Addison-Wesley, 1987.

7. Tanenbaum, A.S. *Computer Networks* (Second Edition). Prentice-Hall, 1988.

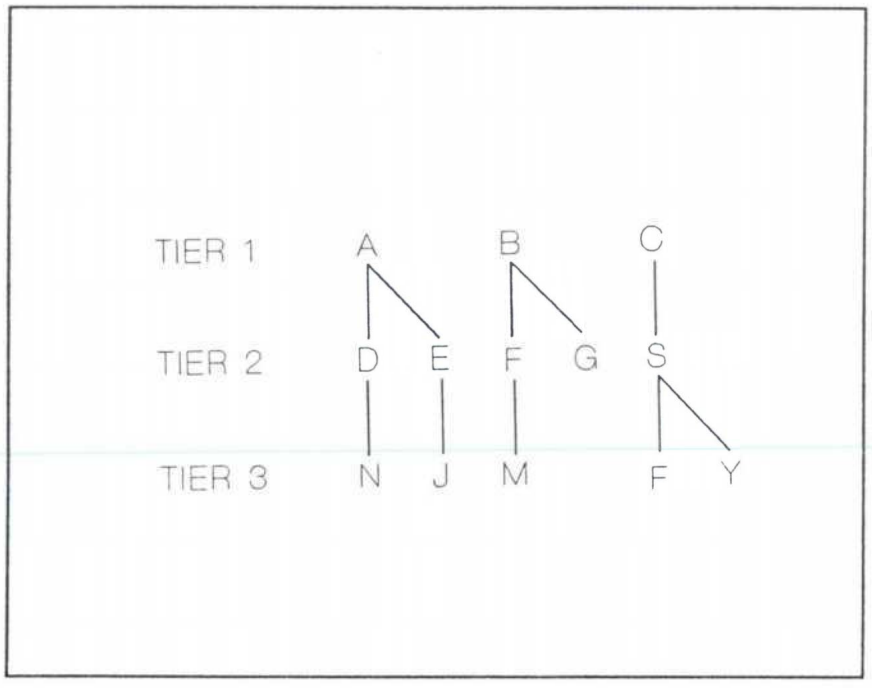


Figure 1. An Example of a Tier Table.

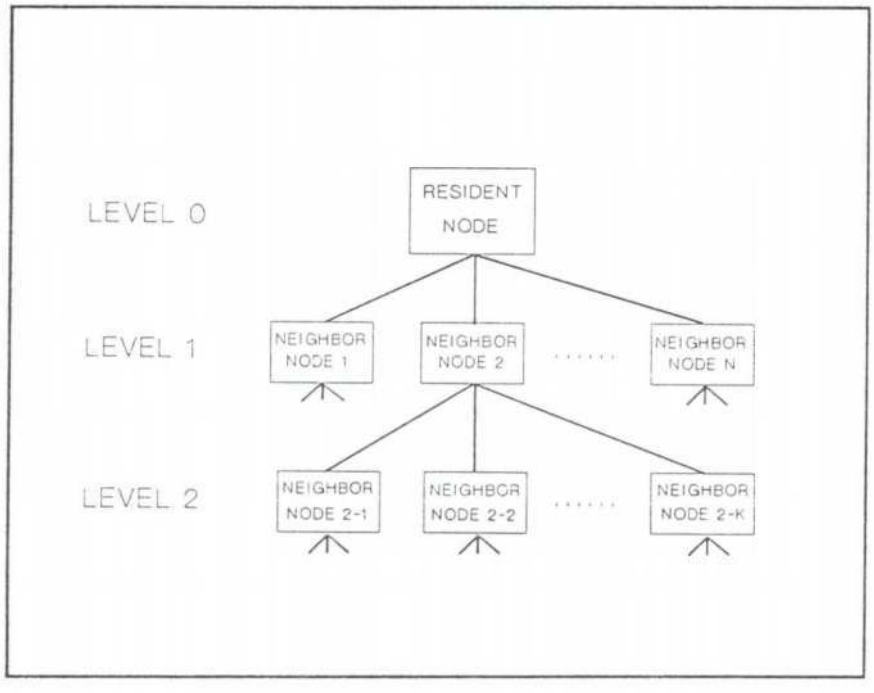


Figure 2. Generalized Routing Table Structure.

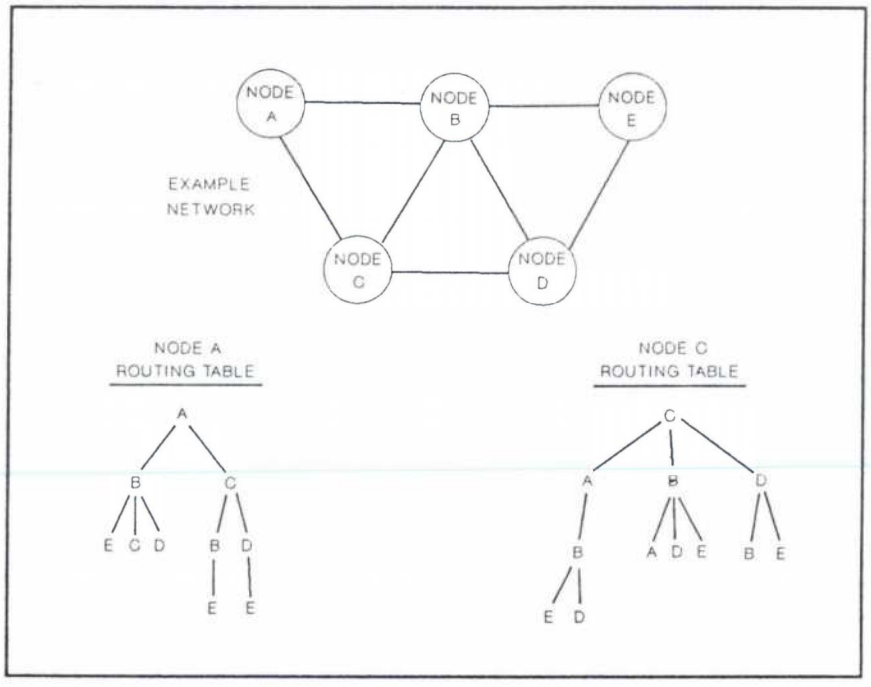


Figure 3. Example of Routing Table Entries.

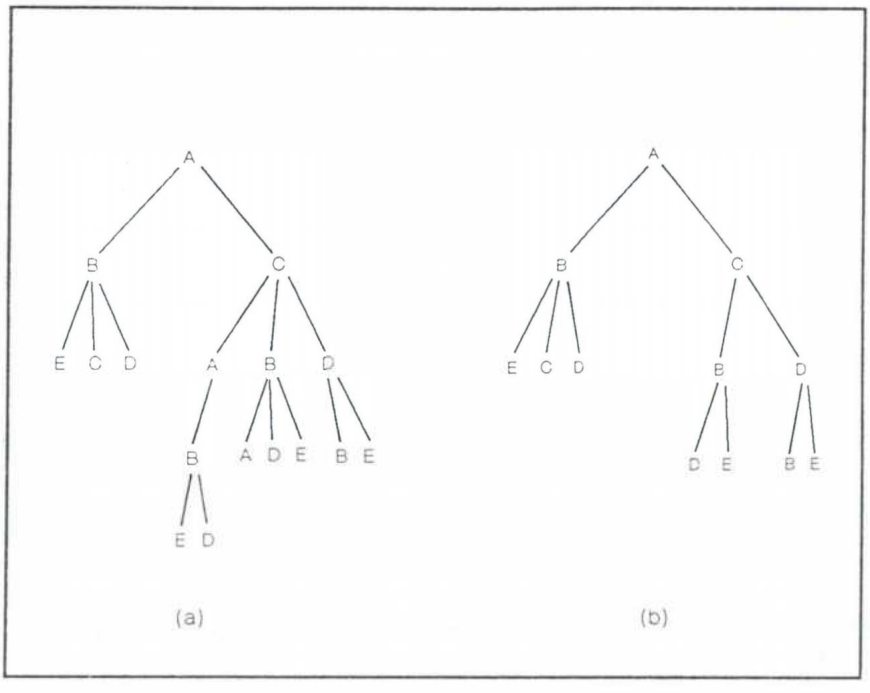


Figure 4. Routing Tables (a) before pruning and (b) after pruning.

Status Report on the KA9Q Internet Protocol Package for the Apple Macintosh

by
Dewayne Hendricks WA8DZP
Doug Thom N6OYU

Introduction

This article describes the current status of the implementation of the KA9Q Internet Protocol Package which we performed upon the Apple Macintosh family of personal computers. The unique Macintosh user interface and its proper utilization by the KA9Q software has been our major objective in the three years since we have started this project and has proved to be quite a challenge. We hope that the users of our implementation have been pleased with the results to date. As of today, there are at least 300 copies of the package out in the world which we have distributed ourselves. We are certain that there are many more copies out there which we don't know about. This is a good deal more than we expected to see three years ago. After all, the Macintosh is not the personal computer that is most linked with amateur packet radio due to its cost when compared to IBM PC's and its clones.

NET

Our work on the NET version of the KA9Q package has been described in some detail in previous articles last year. The first [1] appeared in the 8th CNC proceedings and the second [2] in *73 Magazine*. We will not repeat any of that description here and refer any interested parties to those articles. The version described in those articles (Version 2.0) was released to the public at the Dayton Hamvention earlier this year. Those interested in the availability of that version refer to the notice at the end of this article.

This version is based upon the last version of NET released by Bdale Garbee N3EUA known as Version 890421.1. It also includes several features and facilities from NOS. For example, the Routing Information Protocol (RIP) which appears in the current version of NOS was included in the 2.0 Macintosh release in order to allow NOS and Macintosh stations to interoperate and share routing information in those areas of the world where both packages are in service in the same geographical area.

For those of you into the Macintosh graphical user interface (GUI), you will notice that although there is support for multiple session windows, the windows do not support the familiar 'scroll bar'. The lack of proper scroll bar support comes as a

result of our decision to stick with the THINK C stdio library. This library comes with the compiler and is really just there to provide standard C stdio library functions. No attempt was made by the THINK programmers to provide an environment that would allow the development of standard Macintosh applications with scrollable windows. At best you could say their goal was to provide a way to port UNIX applications to the Macintosh fairly quickly. To fully implement scrollable windows would have meant abandoning stdio and rewriting the entire KA9Q package from scratch in order to restructure the code into a form which would have allowed us to build a standard Macintosh application where all the event-loop logic would have been in the application and not stdio. As were we trying to adhere to Bdale's goal of having all of the various ports of the KA9Q package have a common distribution mechanism, all of our mods were done as IF DEF's. Abandoning the stdio approach would have meant that we couldn't have conformed with Bdale's approach and would have meant that we would have had to do a great deal more work to get a port out the door with the features we wanted. We therefore decided to go with the stdio approach and no scrollable windows.

The mail application (BM) has been converted to THINK C version 4.0. NET however, continues to be based upon THINK C 3.0. We attempted to convert NET to 4.0, but found that the THINK people had left out a number of functions which we had used in version 1.0 of our package. Close study of the new environment showed that if we indeed wanted to convert to 4.0 then we would have to reimplement those missing routines on top of the new stdio. We decided to put off such an effort until we were ready to use the new object oriented class library features introduced in THINK C 4.0

We would like to point out that we found a number of errors in the THINK C 3.0 stdio library. These errors only effect the operation of NET. We have developed a new version of the library called stdio.n which we will provided upon request to those of you who wish to produce a working version of the application from the source.

We plan to put out a maintenance release for version 2.0 sometime during the fourth quarter of this year. This release will include all of the bug fixes which we have made during the course of the year. It will also include some minor feature enhancements, such as new options on some of the commands such as the IP and AX25 Heard commands. It will be available as noted below.

NOS

At this time, we have no plans to do a port of NOS to the Macintosh. The main reason for this decision being that we don't feel that the size of the effort required to do so will justify the returns. We know that this may disappoint a number of you out there, but bear with us as we do have what we feel are good reasons.

NOS as it stands now is fairly machine independent. Karn has spent a good deal of time and trouble to make it so. It is a much better platform to build upon then NET ever was. In order to see how easy it would be to port it to the Macintosh, we did a port in early '89. It took about two weeks of off and on work to get it up, and we encountered no major problems along the way. The biggest piece of work was to remove Karn's new multitasking logic and adapt the code to coexist with MultiFinder. However, this left us with the same problem that we had with NET, no scrollable windows!! We made the decision at this point that any further efforts on our part with NOS would mean that we would have to bite the bullet and do the major restructuring of the code that we had put off with our NET port. This would have been our plan until we remembered the other article we wrote for the 8th CNC [3] which had to do with the need for applications to drive further development in amateur packet radio. Porting NOS as we wanted to do it would have meant doing all of that work and then ending up exactly where we were before, with no applications (make that no Macintosh applications!). Hence our decision to put our efforts into the approach outlined below rather then doing a port of NOS to the Macintosh.

Anyone who is interested in porting NOS to the Macintosh can use our NET work as a point of departure. With a bit of additional effort you should be able to get a version of NOS up which has the same user interface (no scrollable windows) as did NET. We would be happy to provide assistance to anyone so inclined.

The Future

Our present goal is to provide a way to run Macintosh applications over a TCP/IP network in much the same fashion as is done now with AppleTalk. Our view is that what people who use the Macintosh want is to be able to run applications and access data as they normally do. If we can develop an environment which can do that over packet radio, then we will be a lot closer to making those amateurs who use the Macintosh for packet radio work that much happier (ourselves included!).

We are presently working on code which builds upon the AppleTalk Manager in the Macintosh Toolbox which sends and receives AppleTalk (AT) packets over packet radio. We plan to support both raw AppleTalk packets over the air and AT packets encapsulated in IP packets. We hope to report on that work at the 10th CNC next year.

Acknowledgements

We would like to thank all those of you out there who have sent in suggestions and bug reports during the last several years. In particular, we would

like to acknowledge the efforts of **Robert Taylor** KA6NAN, **Edwin Kroeker** N1EWB, **Marsh Gosnell** AD2H and **Adam van Gaalen** PA2AGA. They have all contributed code which was integrated into the final package. In particular we would like to acknowledge the cooperation and assistance of **Tetherless Access Ltd.**, without which this effort would not have been possible.

Package Availability

The object code package is available for a $5 donation (source code for an additional $5) from Doug Thom N6OYU, (c/o Tetherless Access Ltd., 1405 Graywood Dr., San Jose, CA 95129-4778) to cover the costs of the diskette and mailing. The package may also be downloaded via the Internet using 'anonymous ftp' from apple.com, in the directory 'pub/ham-radio' or from the N6OYU landline BBS at (408)253-1309.

References

[1] Hendricks, Dewayne WA8DZP and Thom, Doug N6OYU. "KA9Q Internet Protocol Package on the Apple Macintosh". Proceedings of the 8th ARRL Computer Networking Conference. Newington CT: ARRL, 1989, pp. 83-91.

[2] Thom, Doug and Hendricks, Dewayne. "TCP/IP for the Macintosh". 73 Amateur Radio. October, 1989, pp. 68.

[3]Taylor, Robert KA6NAN and Hendricks, Dewayne WA8DZP. "Application Software for Packet Radio".Proceedings of the 8th ARRL Computer Networking Conference. Newington CT: ARRL, 1989, pp. 210-215.

Texas Packet Radio Society Projects : An Update

Greg Jones, WD5IVD
Tom McDermott, N5EG

Texas Packet Radio Society
P.O. Box 50238
Denton, Texas 76206-0238

Abstract

The last paper published in the ARRL Networking Conference concerning TexNet was in 1987 (1) and we felt it was about time to publish an update. This paper will discuss the current status of the Texas Packet Radio Society's TexNet development, TexNet TexLnk software for the TNC, and the TexNet CARDINAL project. We hope this paper will serve as a method to further distribute information regarding TPRS projects and be able to generate correspondence from interested parties about our activities.

Introduction

Figure 1 shows the current status of the Texas/ Oklahoma/Arkansas network. The Texas and Oklahoma networks were joined in May 1990, adding an additional 400 miles of RF network. The future looks bright for further development and expansion of the network using both RF and landline links. Handling day to day problems on the network has now turned into a part-time job. Ever since the eighth network node, the network is in an ever changing state — nodes will break, get struck by lightening, fall off buildings, and about everything else you can think of will happen! The fun part of the TexNet network was in the designing, implementation, and deployment. Now an enormous amount of work is done in supporting node owners, keeping hardware working, and making the network operational. Building a network is fun, maintaining it becomes an effort of love. Many of the projects described in this paper are the results of now having a network large enough to finally test how TexNet works and operates.

Further expansion of the network outside of Texas will be possible by an agreement recently made with SouthWest Network Services (SNS) for circuits to cities in Texas and other states. TPRS is the sole contact and coordination will occur through TPRS. Please do not contact SNS about this service. These are 9600 bps, point to point circuits, and TPRS intends to extend coverage of the TexNet network to such cities having individuals wanting to support and join in the effort. A few cities available outside of Texas are Oklahoma City, Tulsa, Baton Rouge, New Orleans, Birmingham, Atlanta, and Nashville. These circuits are available now. Future cities include Denver, Los Angles, San Francisco, Phoenix, and others. Those interested in SNS coordinated TexNet service should write TPRS. The possibility of extending the network coverage and allowing this service to act as a bridge to other TexNet-like networks is exciting (3). Also, the development of GLNET (2) in Michigan has been astonishing, occurring at a rate of about twice the Texas network expansion. Since Dayton '90, 11 nodes are now operational. GLNET is currently the third working TexNet network outside of the state of Texas.

Software Update

TPRS has continued refining and improving the TexNet software, with the final version of 1.5 being released this last June. TPRS has been utilizing version 1.4 of the TexNet code for 2 years, and predecessor versions from 1.0 over four years. Versions 1.0 and later provided automatic network routing, with automatic alternate routing. The routing function automatically builds the complete network map in a distributed fashion, as has been reported

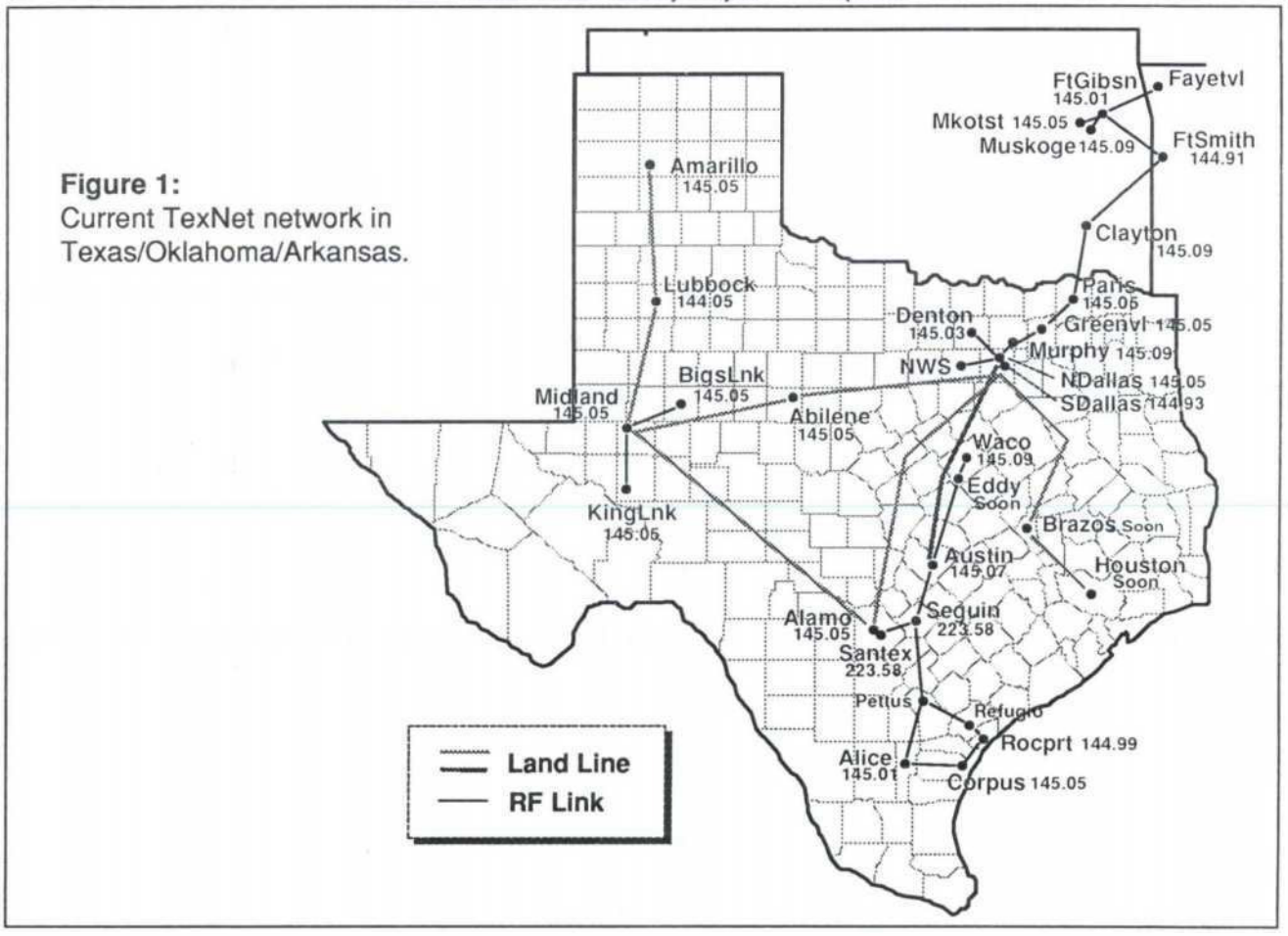


Figure 1:
Current TexNet network in Texas/Oklahoma/Arkansas.

previously (1). Over the last five years, the network has grown to 40 nodes, utilizing 9600 baud 440 trunks, with over 1000 miles of radio paths at 9600 baud and over 1500 miles total networking including landline connections. RF links in the entire system are on the same frequency (+/- 2 PPM). Several links are 70 miles long without the benefit of mountains. During the last few years, we have observed a number of 440 MHz propagation openings. The original design of the network software allowed programming of marginal-trunk lock-outs in each node image, to prevent slightly over-the-horizon conditions from causing routing events to bypass adjacent hops. Approximately ten 600+ mile propagation openings have occurred each year where the network routing has assumed adjacency between pairs of nodes 600+ miles apart. When the band opening ceased, the routes would then be 'stale' and the time it took for the new routes to be corrected was unacceptable. Another affect of band openings and node failures was the creation of asymmetrical routing in the network, which is seen by having a good 'to' route but the 'return' route has failed, causing packets to be lost in the either.

In late 1988 the TexNet Network Management System (NMS) was placed on the air to take care of wide-area network anomalies and manage day to day operations of the network. Creating this system both supported the short term problems in wide-area network support and allowed the long term development of an intelligent network supervisor. The NMS 'polls' each node in the system every 10 minutes, and calculates network statistics. Each night at midnight, it uploads three network performance reports to one of the network message servers. These reports show the retry rate, transmitted frame count, received frame count, buffer utilization, and number of user log-ons on every link on every node in the network for the day. Additionally, the outage-table shows each ten-minute slot during the day for each node, showing information on which particular nodes did

not respond to a poll (within 59 seconds). This information has been used to show time-of-day dependencies on network performance. Additionally, the NMS takes automatic corrective action at nodes that appear to be dead (don't respond within 59 seconds of a query). It is capable of sending a node-specific 'fire-code' sequence to fail-safe hardware in each node by broadcast-flooding this sequence throughout the network in order to remotely reboot any node. No operator intervention is required for any of these events, thus making the network more reliable.

On February 17th, 1990, the TexNet Support Group met to discuss the future of TexNet software and to discuss the anomalies seen in the network. This meeting has led to the current 1.6 software now under development and testing. Version 1.6 is aimed strictly at correcting wide-area network operation anomalies which have only now been discovered with the network of this size.

The new version 1.6 code fixes affected network trunk conditioning under extremely marginal signal conditions, but more importantly, allows remote manipulation of the network routing tables. This new network code allows the NMS to read and reprogram the network routing tables from a central location without any operator intervention. The new version of the NMS that supports this will allow for NMS to realign the network routing automatically to compensate for propagation anomalies, or certain types of equipment failures. The network is still capable of automatically setting up its routes in a completely distributed fashion, but we can now offer over-riding 'help' for the tables when necessary.

TexNet TexLnk TNC Project

The original TexNet code was developed on the TNC II, until the first TexNet NCP (4) was built and the code moved to the NCP. After a year of working on the 1.4 and 1.5 code and examining the originally developed code, Jim Brooks, W5ERO, was able to get a single port version of TexNet code again operational on a TNC II. The project is now called TexLnk, since the purpose of TPRS getting this software working again was to create inexpensive high-speed links between network nodes and provide thin-route networking in West Texas where the population of packet users does not demand full-blown TexNet nodes yet. The project was not done to create another, poor operating, single frequency network. TexLnk and CARDINAL, described below, are both intended as supplemental technology to allow more robust networking options.

During the beta-test period, two high-speed linking nodes were deployed and have worked as expected, providing 9600 baud networking between sites with NCPs. This is accomplished by adding a TPRS 9600 baud modem as the external modem. One drawback to the current TNC version is that it does not support the hardware 'firecode' feature available on TexNet NCPs. Firecode allows the node to be hard-reset remotely by command. One of the nodes was equipped with a tone decoder reset and the other was not. The one without required a 35 mile drive to reset it, which has only been required once, since most problems can be corrected by a soft-reset which will still work without the hardware 'firecode' circuitry.

TPRS has been hesitant in releasing this software to the public, since TPRS feels that people will try to make it work as a single-frequency network, which will work, but not give the same performance as TexNet. TexNet works well because: 1) the network trunks, which are on a separate frequency to avoid congestion problems, operate eight times faster than user access to the network, 2) the network software works very well, and 3) the quality network services provided to the user. TPRS is releasing this software as a 'SHAREWARE' (5) product. Contact TPRS about getting a copy of the software and utilities. A manual, image editor, and paper on how 'not to use it' should be available by the networking conference from TPRS.

TexLnk provides the same software functionality as a full-blown TexNet node, except that it is a single port node. The features provided are : Digipeating, Conference Bridge, Network Interface, and Local Console. For more information on TexNet services refer to the TPRS TexNet User's Manual (6) or the 73 article on TexNet published in October 1989. (7)

Two dual port projects are being worked on. Hopefully an inexpensive dual-port solution will be available in the coming months to allow this software to be used as a low-cost dual-port TexNet node running on a TNC II, providing 1200 baud local access and 9600 baud networking.

CARDINAL Project

The CARDINAL card is a communications coprocessor card for an IBM-PC XT/AT compatible computer. The design has been tested and is operational with boards in layout. CARDINAL allows vastly simplified development of AX.25 based applications, network applications, and additional services or features for the TexNet network. The original intent of the CARDINAL project was to replace the first and only software development platform for TexNet (8). By having a plug-in card, we can now use the power of the PC for development and provide multiple cards to TexNet developers for faster code development. From the initial design, the current project has been worked on to allow the card to do much more.

CARDINAL contains almost identical components to the TPRS Node Control Processor (NCP) board that is used as the standard TexNet node. It contains 2 SIO/0 serial chips, a Z80 processor, Z80-CTC, 64K of RAM, a dual-port connection between the Z80 and the PC-XT bus for use of the RAM, and optional hardware breakpoint logic. The dual-port is actually just a standard single-port static RAM, but special logic on the board 'steals' the Z80 refresh cycles, and sequences the PC access during this time. This is a very cost-effective way to implement the dual-port hardware. The current TexNet code image runs, unaltered, on the PC-plug-in card. For the Z80, layer-2 drivers for versions 1 and 2 of AX.25 are available (for those who don't really care about TexNet) and a native TexNet layer-3 driver is developed and being debugged. All three have been proven operational at 9600 baud.

The PC drivers allow the passing of data frames, and control requests from the PC to the Z80 processor and vice versa. The Z80 acts as a coprocessor, and in fact, operates continuously even as the PC is rebooted with a CTRL-ALT-DEL, since the CARDINAL card ignores the PC-reset line. These drivers are supplemented by a TSR in the PC that periodically queries the Z80 for completed send and receive packet activity. The TSR runs in the background, and will support shared ring-buffers for both transmit and receive in the PC RAM so that the application on the PC can ignore any real time aspects of the communication task. An application has been written in Borland's Turbo-C that merely queues up transmit requests or examines received queues for incoming data. In this way, the PC application task needs to know almost nothing of the AX.25 or TexNet protocols, and have no timing constraints whatsoever. A simple network-capable multi-user read-only BBS with directory functions took less than 3 pages of 'C' code to implement using these drivers. Since the function calls and buffer passage resolve all data and control possibilities via function codes, it is not necessary to parse ASCII strings to manage connection setup and teardown.

Extensive debug utilities are written, that allow download of coprocessor software, single-step, breakpoint, register alteration, programmable trap, memory dumps, and disassembly of the current image. The debugger is written in 'C' and runs foreground on the PC, but can leave hardware traps and breakpoints armed before exiting. These armed traps allow debugging both the coprocessor and simultaneous applications on the PC.

The low-level device drivers on the PC are based on 'handles' so that the actual I/O addresses and the number of CARDINAL cards plugged into the PC can be easily selected by function calls. The device driver returns a 'handle' to the selected CARDINAL card. This facilitates changing the actual PC I/O address of the card, and allows multiple cards to be plugged into a single PC. The current card has up to 4 synchronous channels and straps on the CARDINAL card allow 4 different addressing ranges, resulting in the ability to construct a 16-port (synchronous AX.25) PC configuration. The TSR level drivers have programmable transmit and receive buffer ring sizes, allowing optimization for different applications. The PC drivers and TSR are written in 8086 assembly language.

The native layer-3 interface to TexNet makes it fairly easy to add new and diverse network capabilities to TexNet. Applications being planned include support for other network types, database applications, DX Cluster support, a new weather server, and others. Having the application in the PC and not in the node, will allow us to get away from the current space constraints and having to write Z80 assembly for the TexNet NCP.

An example usage of CARDINAL might be : a user at any node types the string: 'Message @ CARDNAL' and is automatically connected to and assigned a unique network-layer port number at the CARDINAL card. The network knows how to route the request (due to it's automatic routing function) and the recipient node knows the entire layer-2 and layer-3 address from the network datagram structure. Up to this point, the PC software hasn't done anything yet! The PC software is passed a buffer with the user connect call string, and the unique port and layer assignments by the Z80. This simple scheme facilitates adding new applications that appear to be part of the network, but without new users needing to know anything about callsigns, locations, or network topology, etc. This maintains the transparent appearance to the user which is part of the TexNet network design. The 'C' application on the PC responds with either a log-on function, or an application selection query message to the user, depending on what the user has asked for. Plans are being discussed to add a services broadcast to version 1.6 which would allow users to know which services are available at which nodes on the network.

Conclusion

What comes after these projects for TPRS? Only time will tell, but some guesses are: providing more diverse services across the network, increasing network throughput either by increasing speed or by using data compression — allowing users to access the network faster, integrated voice and data networks, TexNet ported to other platforms, and probably a whole bunch of other brain storms we haven't even had yet. If you would like to know more about any of the above projects or want to find out more about TPRS, please feel free to write.

NOTES:

1. McDermott, Tom, N5EG. "Overview of the TexNet Datagram Protocol". Proceedings of the ARRL 6th Computer Network Conference. Redondo Beach, California. 1987.

2. Great Lakes Network. Contact Jay Nugent, WB8TKL, 3081 Braeburn Circle, Ann Arbor, MI, 48108.

3. "SouthWest Network Services". TexNet Support Group, TPRS Quarterly Report. Volume 7, Issue 1, July 15th, 1990. p 18.

4. Node Control Processor. The computer section of a TexNet node.

5. SHAREWARE - If you use it, then please send in the registration per-node to TPRS so that we can keep working on new things.

6. TexNet User's Manual available from the TPRS for $1.00.

7. Jones, Greg, WD5IVD. "TexNet Packet-Switching Network" 73 Amateur Radio. Issue 349 October 1989.

8. The TESTBED node was the only memory resident TexNet node built. Built by Tom Aschenbrenner, WB5PUC, it was used to write all of the original version 1.0 code. TESTBED was an elaborate multiboard homebrew CPM system which emulated a TexNet node.

FlexNet

THE EUROPEAN SOLUTION

Gunter Jost, DK7WJ
Lichtenbergstr. 77
D-6100 Darmstadt
DK7WJ @ DB0GV.DL.EU

Joachim Sonnabend
Annastr. 4
D-6082 Moerfelden
DG3FBL @ DB0GV.DL.EU

FlexNet-Group
Bunsenstr. 2a
D-6100 Darmstadt
Fax: 49 6151 895718

This paper describes the design and implementation of FlexNet. This is a packet switch software with a complete new design made in Germany. It is beginning to become the standard in Central Europe. Running on a dedicated hardware with an unique design it is easy to put cheap and reliable packet switches on our mountain sites.

1. Introduction

In the last years of increasing Packet Radio activities, some groups in Germany began to develop hard- and software for network nodes. In the Frankfurt area the RMNC (Rhein Main Network Controller) was developed by a small group of people. This is a nodecomputer, optimized for high data thorughput and an extreme good price/performance relation. Parallel to the hardware, an absolute new packet switch software was developed in Darmstadt. After only 15 months of distribution, this software is running on more than 100 RMNCs in DL and the neighbour countries. A portation of the system to other hardwares is planned. RMNC/FlexNet is one of most modern, fastest and on the same hand cheapest nodecomputer concept available in Europe. A lot of nodes change to the RMNC/FlexNet, because their old packet switches have reached the barriers of their efficiency or an extension would be too expensive. On the other hand there is a built-in autorouter in the newest version of FlexNet, which is completely new developed and has been proofed to be extremely efficient.

2. Network Topography in DL

For the understanding of the FlexNet layout we want to explain some aspects of the PR situation in DL. The frequency assignments and the goverment regulations had a lot of influence on the development of our PR network. Network nodes and mailboxes which are operated unmanned need a special license. This has a lot of disadvantages which we don't want to discuss in here. But there is a kind of coordination which is an

advantage, because we realized a relative good structurized network. Fortunately there were no restrictions in using new protocols. By this reason it was possible to create several interesting projects for digipeaters and network nodes. Today there are more than 100 "official" nodes in our small country. Because of the frequency situation most of them only have one user channel (mostly on 70cm). The nodes are connected with exclusive links on 23cm. This is the reason why these lines are free of collision, undisturbed and why it is possible to reach a high performance with a relative low datarate. Lots of lines are still working with 1200 Baud half duplex, but mainly the most frequent lines are changed to 9600 Baud full duplex. Because of the limited frequency assignments, higher data rates are hard to realize. This is the reason why we are thinking about changing to the microwave bands. The links have a length of 80 - 200 km and the nodes have about 2-6 links to othes nodes, each on an exclusive frequency pair. For this a huge effort of RF components is necessary. This explains once more the call for a cheap nodecomputer with great performance data.

3. Hardware: RMNC

The RMNC is a modular system. All boards have eurocard format (100 * 160 mm). They are connected with a 64 pin backplane for parallel data exchange. To make it cheaper, all components which are common for all channel processors are seperated on a so called reset-IO board. This board has a reset generator with watchdog, a clock generator and 32 switch IO's for remote controls. The channel processor has already been developed in 1986 and contains the following components:

- CPU MC6809 with 4 or 8 MHz clock
- VIA 6522 for buscomunication
- SCC 8530 for HDLC-IO
- 27256 EPROM 32kByte
- 2 * 43256 RAM 64kByte, one with battery backup for parameter storage
- TCM 3105 Modem for 1200 Baud, optional
- Modem disconnect plug for external modem

The boards are connected with a backplane which is availible in the surplus trade as an european industrial standard. The whole computer is supplied with 5 volts only. The boards are distributed as kits. However, the board layouts are in the public domain for noncommercial use. If you are able to get the components quite cheap, the price for the whole RMNC including backplane, case, power supply and 6 channels could be under $350.

4. Software: FlexNet

FlexNet was developed out of classical digipeater concepts. The first RMNC-Software (Ver. 1.x) had a list of its neighbour digipeaters which was coded in EPROM. So every digipeated frame has been sent to the right channel. This made it possible to manage exclusive links. The aim for the development of the first FlexNet version was to get the connectability of the node and a Hop-To-Hop-Acknowledge with simulated digipeating. With this

concept the disadvantages of the Net/Rom-concept, which spred around in DL, should be avoided.

FlexNet was developed without third party source code. The basics for the development was the AX.25 protocol as well as monthlong brainstorming with friends. Thanks to the programming language C it is possible to port the program to other computers. Because of the bad performance and the inefficient code we did not port it to the Z80-TNCs.

The first FlexNet implementation on an RMNC came in the beginning of 1989 with the version number 2.0 to make a distinction to the old RMNC software. The 2.x version and the new version 3 have the following features:

- Connectability, the user gets a Ctext
- Callable infotext (Help, Info, News)
- All parameters, activity- and link informations are readable
- Conversation mode for roundtable discussions
- Complete remotecontrol for the Sysops (links, baud-rates, parameters, infos, beacons), all commands in plain text
- There is only one callsign for the whole node, made by the master-slave-principle. The routing to the different channels is made by analyzing the next callsign (linklist) or by SSID, if there is more than one user channel
- Users cannot access so-defined link channels directly, therefore the traffic on the links is collision free
- Simulated digipeating with the known scheme
  "Connect ‹User› via ‹Node› [,‹Node›]"
  UA is delayed until the connection over the network is made
- Flow Control with RNR/RR independant for each hop
- Window size for QSO is equal to the sum of the maxframes of the involved nodes, as a result good throughput on long range connections
- Selective Flow Control for each QSO made possible by virtual circuits, so congestion cannot occur for fast links when slow links on the same node have problems to get rid of their frames
- Round-Trip-Timer replaces FRACK parameter
- Errormessage on breakdown of the connection:
  "‹Node›: Link failure"
- No timeout for QSO's via the node

There were a lot of improvments made on maintanance. On general demand we built in a connect command. This is done without changing the SSIDs. Behind the node processing the C-command, the same path will be created like it would have been with normal digipeating.

Example: ‹User› to ‹Node›: "C ‹Friend›"
The node works to the other side with the callfield
‹User› -› ‹Friend› via ‹Node›*

With this trick our friend always can connect back to us without the need for us to tell him the path to use.

5. FlexNet Autorouter (FlexNet V.3)

The autorouter of the FlexNet V.3 is a complete new development. Because of the bad performance of the known routers we decided not to make it compatible with them. Wo did not like the long node lists where only a few nodes where really accessible. And we wanted to build a router which is able to adapt himself quickly to changes in topograhy and failed or overloaded links. And of course it should measure the actual availibility and performace of the links so that it is really able to find the best way to any known destination. The development started in the beginning of 1989. The first tests with several nodes were made in winter 89/90. In August 1990 the router was released with FlexNet V.3.

The linklist (list of the neighbours) will be put in manually by the sysops. (An automatic detection of neighbours by using broadcasts would be possible, but we don't think its useful. In Germany we don't need it because of our fixed link concept).

Internode communication only takes place between neighbours who both have their partner in the link lists. By this unwanted nodes can easlily be locked out.

There is a permanent L2-connection between the nodes, by this we can exchange destination infos and are able to test the functionality of the links. So it is easy to detect a breakdown of a link or of a neighbour and the router is able to react immediately.

Every 5 minutes FlexNet makes a runtime measurement for a 200 Byte testframe within the internode connection. With the last 16 measurements the average runtime to the neighbour will be calculated. With this calculation the line utilisation and the influence of retries will be recorded. This time is used for routing and the users will be informed about it in the linkinfo. The best way to a destination is the route with the smallest actual runtime calculated by adding the link runtimes.

Links to neighbours which cannot handle the internode protocoll (i.e. mailboxes) are checked with periodical connect tries. This also makes it possible to calculate the runtime and to detect a breakdown after some tries without success. An UA or DM is sufficient as a response, so mailboxes should lock out the call of their local node.

The so-collected destination infos are exchanged with all FlexNet neighbours. Because of the internode connection no periodical repetition is neccesary. When the runtime to a particular destination changes significantly, the info is updated. By this the network can quickly react to any changes. When a link breaks, the internode qso goes down after a few minutes. Then all routes via that link are deleted immediately.

On each node the user can ask for the runtime to each known destination, and if he is interested in how the router works, he can ask for the complete path to a destination. To get this

information the node needs some special internode traffic because each node only knows the next downstream neighbour. Therefore the path info may be delayed for some seconds.

By the use of an adaptive hold down timer bad destination news are growing faster than good ones. Because unreachable destinations and unusable links are quickly detected, the destinations shown in the lists were really available a few minutes ago.

Sink-Tree-Algorithm with adaptive Hold-down-timing guarantees loop free routes. There is no necessarity for a Time-to-Live-Counter.

Virtual Circuit: There is no necessity of a Layer 4 with End-to-End-Acknowledge. One step connection all over the whole network is possible, the user doesn't have to adapt himself. There is no need for an 3-step up/cross/downlink setup, however users can do that if they like it. Communication establishment with C-commands or one step with the command:

"Connect ‹Friend› via ‹first node›,‹last node›"

or

"Connect ‹Node› via ‹first node›"

The path is allways reversible because the first node is included in every frame. The connected user is able to recognize the way back, even if the QSO was build up with C-command and some detours. He always sees a frame with the first and last node in the digipeater callfield.

Because there is no L4 it is not necessary to use any fragmentation on long I-frames. The whole routing is done in the AX.25-digipeating field. We got it with AX.25, and now it is very sensefull to use it for routing.

At every node on the way to the destination it is possible to get the data of the QSO's and with these data (poll or reject states, unacked frame counts) it is easy to find bottlenecks in the network (and it's fun to look around what's happening). The overhead by transmitting the entrance and exitnodes in the digipeater field is not larger than at existing datagram protocolls (L3/L4-header).

It is im portant for other protocols and experiments that FlexNet routes everything: Frames that do not belong to a running AX.25 QSO are routed as datagrams (i.e. UI's), when an SABM is seen, a virtual circuit is built automatically. In both modes the PID's will be passed through unchanged. So all modes including TCP/IP can use FlexNet and draw advantage from the autorouter.

6. Performance of the actual RMNC implementation

Today maximum 9600 Baud fullduplex/channel, extension to ≈64kBaud possible and planned
16 channels/node
More than 100 QSO's/channel, ≈200 QSO's/node.
Maximum data transfer rate of the whole node is ≈500kBit/s
570 destinations storable
20 links (neighbours)

The Master EPROM with the parameters is generated by a supplied parameter compiler running under MS-DOS. The input is a text file with the same syntax used for remote control. No patches are neccesary. Slave processors allways have the same code, parms are downloaded from the master processor after reset.

7. Future development

With the 6809 we have reached the limit of the FlexNet V.3 by means of code length and data space. Because of the low costs for a FlexNet node we didn't think about a larger computer until now. FlexNet's reputie is to be very fast and efficient so we don't think about porting it onto slower computers with lower data throughput. Nevertheless it is very easy to port the software to any other computer because allmost the whole code is written in ANSI C.

In the meantime we are discussing about porting the code to PCs with plug-in HDLC boards, however that might be a more expensive solution without performance gain.

Therefore we are developing a new channel processor for the RMNC, which will have more space for code and data and which will be able to handle faster links. It should be integrated into the existing RMNC's to save the investments already made.

The technical data:
- Processor MC 68302
- 1-4 MB RAM, 512 kB EPROM
- DMA-transfers to the parallel backplane with ≈8MBit/s
- 2 channels/board with max. 1MBit/s
- RS232-port

On that board we can run TCP/IP, S&F functions and lots of other things not yet in mind. For the high level functions it is only needed once in the system as a master processor, however for fast links it can also be used as a slave. Then the data rate on the backplane will draw advantege of the DMA.

8. Conclusion

With RMNC/FlexNet a Packet Switch was created which is optimally adapted to the german network structure. It was lot of work to build a completely new design, but now we know that it made sense. We did it for fun, and we had lot of fun. Especially we feel that it is more satisfying to find self made bugs than scratching around in sources from other people.

Already without the autorouter the users liked FlexNet because the design and handling is easy to understand. The network is fully transparent for other protocols. Now with V.3, users and even sysops no more have the need to look at the network topographie. There is not one parameter for the router sysops can play with, all informations except the neighbour list is automatically built. So users can say "C ‹mailbox› via ‹node›" to their TNC and it will work on the best route available. It is nearly impossible to find a better route to the wanted destination than the router chooses. Sysops like FlexNet because they need no tables with parameter lists, everything can be controlled with plain text commands. Maintenance of the links is easy, the node keeps track with changing link qualities. There is no need for unproto tests to check the links. Even worse, it is senseless to check a link with unprotos like "‹sysop› to ‹test› via ‹mynode›, ‹othernode›, ‹mynode›", because this frame might be routed through some neccesary deviations and come back with the original callfield.

All frames running through the network carry the information about the originator and the first node on which it came into the network. Control authorities think that to be very useful.

In the meantime foreign groups are interested in FlexNet and wo hope to be able to support them. RMNCs are now running in the GDR, Belgium, France, Switzerland, Austria, Italy and Hungary. Especially in eastern countries our cheap concept is of great interest.

9. Acknowledgements

We want to thank the users and sysops in our area who suffered from numerous bugs in the preliminary versions of FlexNet. Sysops have changed EPROMS many times and even had to change the IC-sockets because they were worn out. Lots of detailed bug reports helped to make the code as stable as it is today. We also want to thank the software gurus wo put their ideas to the public domain. The brainstormings during many meetings brought lots of ideas which we only had to realize. We just tried to make it better, but we had an easy job by starting quite late and looking at other concepts.

At last thanks to you for reading this paper although the english might not be very professional. In the near future we will have an english documentation written by an american ham.

Reference: Tanenbaum, A.S. Computer Networks (2nd edition), Prentice Hall: Englewood Cliffs, NJ. 1989

MACA[1] - A New Channel Access Method for Packet Radio

Phil Karn, KA9Q

ABSTRACT

The existing Carrier Sense Multiple Access (CSMA) method widely used in amateur packet radio on shared simplex packet radio channels frequently suffers from the well-known "hidden terminal problem" and the less well known but related problem of the "exposed terminal." This paper proposes a new scheme, Multiple Access with Collision Avoidance (MACA), that could greatly relieve these problems. MACA can also be easily extended to provide automatic transmitter power control. This could increase the carrying capacity of a channel substantially.

1. Introduction

In the classic hidden terminal situation, station Y can hear both stations X and Z, but X and Z cannot hear each other. X and Z are therefore unable to avoid colliding with each other at Y. (See figure 1.)

In the exposed terminal case (figure 2), a well-sited station X can hear far away station Y. Even though X is too far from Y to interfere with its traffic to other nearby stations, X will defer to it unnecessarily, thus wasting an opportunity to reuse the channel locally. Sometimes there can be so much traffic in the remote area that the well-sited station seldom transmits. This is a common problem with hilltop digipeaters.

This paper suggests a new channel access algorithm for amateur packet radio use that can minimize both problems. This method, Multiple Access with Collision Avoidance (MACA), was inspired by the CSMA/CA method (used by the Apple Localtalk network for somewhat different reasons) and by the "prioritized ACK" scheme suggested by Eric Gustafson, N7CL, for AX.25. It is not only an elegant solution to the hidden and exposed terminal problems, but with the necessary hardware support it can be extended to do automatic per-packet transmitter power control. This could substantially increase the "carrying capacity" of a simplex packet radio channel used for local communications in a densely populated area, thus satisfying both the FCC mandate to use "the minimum power necessary to carry out the desired communications" (Part 97.313) and to "contribute to the advancement of the radio art" (Part 97.1 (b)).

[1] MACA is an acronym, not a Spanish word.

2. How CSMA/CA Works

CSMA/CA as used by Localtalk works as follows. When a station wants to send data to another, it first sends a short Request To Send (RTS) packet to the destination. The receiver responds with a Clear to Send (CTS) packet. On receipt of the CTS, the sender sends its queued data packet(s). If the sender does not receive a CTS after a timeout, it resends its RTS and waits a little longer for a reply. This three-step process (not counting retransmissions) is called a *dialogue*. Since a dialogue involves transmissions by both stations, I will avoid confusion by referring to the station that sends the RTS and data packets as the *initiator*, and the station that sends the CTS as the *responder*.

The RTS packet tells a responder that data follows. This gives the responder a chance to prepare, e.g., by allocating buffer space or by entering a "spin loop" on a programmed-I/O interface. This is the main reason Localtalk uses the CSMA/CA dialogue. The Zilog 8530 HDLC chip used in the Apple Macintosh can buffer the 3-byte Localtalk RTS packet in its FIFO, but without a DMA path to memory it needs the CPU to transfer data to memory as it arrives. The CPU responds to the arrival of an RTS packet by

returning a CTS and entering a tight read loop, waiting for the data to arrive. [2] (A timeout prevents a system lockup if the data never arrives.)

As is standard for CSMA schemes, CSMA/CA requires stations to stay off the channel when another transmission is already in progress. CSMA/CA also requires any station that overhears an RTS or CTS packet directed elsewhere to inhibit its transmitter for a specified time. This helps reduce the probability of a collision with a subsequent CTS or data packet. This is the CA or *Collision Avoidance* part of CSMA/CA. However, collisions are not a major problem on Localtalk; the network is physically small, carrier sensing is fairly rapid, the data rate is relatively low, and (if the network is properly built) there are no hidden terminals. Plain CSMA would work well, but there was little extra cost to the CA feature (given that the RTS/CTS dialogue was already needed for other reasons) so it was included.

3. Turning CSMA/CA into MACA

Hidden and exposed terminals abound on simplex packet radio channels, and this makes them very different from Localtalk and most other types of local area networks. When hidden terminals exist, lack of carrier doesn't always mean it's OK to transmit. Conversely, when exposed terminals exist, presence of carrier doesn't always mean that it's bad to transmit. In other words, the data carrier detect line from your modem is often useless. So I'll make a radical proposal: let's ignore DCD! In other words, let's get rid of the CS in CSMA/CA. (It's too hard to build good DCD circuits anyway...)

Instead we'll extend the CA part of what we'll call MA/CA (or just plain MACA). The key to collision avoidance is the effect that RTS and CTS packets have on the other stations on the channel. When a station overhears an RTS addressed to another station, it inhibits its own transmitter long enough for the addressed station to respond with a CTS. When a station overhears a CTS addressed to another station, it inhibits its own transmitter long enough for the other station to send its data. The transmitter is inhibited for the proper time even if nothing is heard in response to an RTS or CTS packet.

Figure 3 shows an example. Station Z cannot hear X's transmissions to Y, but it *can* hear Y's CTS packets to X. If Z overhears a CTS packet from Y to X, it will know not to transmit until after Y has received its data from X.

But how does Z know how long to wait after overhearing Y's CTS? That's easy. We have X, the initiator of the dialogue, include in its RTS packet the amount of data it plans to send, and we have Y, the responder, echo that information in its CTS packet. Now everyone overhearing Y's CTS knows just how long to wait to avoid clobbering a data packet that it might not even hear.

As long as the link between each pair of stations in the network is reciprocal (i.e., all the stations have comparable transmitter powers and receiver noise levels), the receipt of a CTS packet by a station not party to a dialogue tells it that if it were to transmit, it would probably interfere with the reception of data by the responder (the sender of the CTS). MACA thus inhibits transmission when ordinary CSMA would permit it (and allow a collision), thus relieving the hidden terminal problem. (Collisions are not *totally* avoided; more on this point later.)

Conversely, if a station hears no response to an overheard RTS, then it may assume that the intended recipient of the RTS is either down or out of range. An example is shown in figure 4. Station X is within range of Y, but not Z. When Y sends traffic to Z, X will hear Y's RTS packets but not Z's CTS responses. X may therefore transmit on the channel without fear of interfering with Y's data transmissions to Z even though it can hear them. In this case, MACA allows a transmission to proceed when ordinary CSMA would prevent it unnecessarily, thus relieving the exposed terminal problem. (Because modems have a capture effect, hearing a CTS doesn't *always* mean that you'd cause a collision if you transmit, so the problem isn't yet completely solved. More on this point later.)

4. Metaphors for MACA

MACA is not really a novel idea; it merely formalizes a procedure many people (not just radio amateurs) instinctively use in personal conversation. A typical cocktail party has many

[2] It would be nice if we could use this feature on packet radio with our programmed-I/O HDLC interfaces (e.g., DRSI PCPA, Paccomm PC-100). Unfortunately, if our RTS/CTS packets carry full source and destination call signs, they would not fit into the 3-byte 8530 FIFOs. So high speed operation will still require either DMA or a dedicated I/O processor.

simultaneous conversations. The average guest seldom waits for total silence in the room before he speaks, but if someone asks him to pause because he is trying to hear someone else, he will usually do so. The MACA RTS packet is analogous to Bob saying "Hey, Tom!" and CTS packet is analogous to Tom responding with "Yeah, Bob?". This causes most people to stop talking if they are close to Tom (except, of course, for Bob). The same thing (should) happen in manual amateur radio operation whenever a station finishes a transmission with "go only" (or "KN" on CW or RTTY).

The Prioritized ACK scheme also involves overheard packets that inhibit other stations for specified periods of time. In this case, the inhibiting packet is a data packet and the protected station is sending an acknowledgement that may not be audible at the inhibited stations. Full protection against collisions is not provided (data packets can still collide) but the performance improvement due to the lower ACK loss rate is reported to be substantial.

More formally, MACA can also be seen as a single-channel, time-multiplexed form of Busy Tone Multiple Access (BTMA). In BTMA, receivers transmit "busy tones" on secondary channels whenever their receivers are active. This warns the other stations in range that they should not transmit even if they hear nothing on the data channel. On the other hand, stations not hearing busy tones are free to transmit even if there is already a signal on the data channel. Indeed, stations need not pay any attention at all to the data channel when deciding to transmit; only the busy channel matters. As long as the propagation characteristics are identical between the main and secondary (busy tone) channels, BTMA is effective. Unfortunately, the need to use widely separated frequencies to avoid self-interference tends to make the link characteristics less symmetrical. BTMA also obviously increases the hardware complexity and spectrum requirements of each user station. On the other hand, because MACA uses the same channel for the "busy tone" and data, paths between pairs of stations are much more likely to be symmetrical.

5. Collisions in MACA

Unlike BTMA, however, collisions between RTS packets can still occur in MACA. These are minimized with a randomized exponential back-off strategy similar to that used in regular CSMA. Since there is no carrier sensing in MACA, each station simply adds a random amount to the minimum interval each station is required to wait after overhearing an RTS or CTS packet. As in regular CSMA, this strategy minimizes the chance that several stations will all jump on the channel at the instant it becomes free. The extra random interval would be an integral multiple of the "slot time", and in MACA the slot time is the duration of an RTS packet. If two RTS packets collide nonetheless, each station waits a randomly chosen interval and tries again, doubling the average interval on each successive attempt. Eventually one of them will "win" (i.e., transmit first) and the CTS from its responder will inhibit the "losing" station until the winning station can complete its dialogue.

Even though collisions can occur between RTS packets, MACA still has the advantage over CSMA as long as the RTS packets are significantly smaller than the data packets. As long as this is true, collisions between RTS packets are much less "costly" than the collisions that would otherwise occur between data packets. The savings in collision time also pays for the overhead of the RTS and CTS packets.

As mentioned earlier, the basic MACA protocol only reduces the chances of collisions involving data packets; it does not guarantee that they will never occur. This is because a CTS packet requires a certain minimum signal-to-noise ratio at a station for it to be understood and obeyed. Even if the station powers are well matched, a pair of stations might have just enough of a path between them to allow them to interfere with each other's reception of weak signals, but not enough of a path to allow them to hear each other's CTS packets. Although the seriousness of this problem is unknown, it does appear that the power-controlled version of MACA discussed later would greatly reduce it.

6. Bypassing the MACA Dialogue

If the data packets are of comparable size to the RTS packets, the overhead of the RTS/CTS dialogue may be excessive. In this case, a station may choose to bypass the normal dialogue by simply sending its data without the dialogue. It must, of course, still defer to any RTS or CTS packets it may overhear.

Of course, the bypass mechanism carries with it the risk of a collision. However, for some types of data packets this may be an acceptable tradeoff. An example might be the acknowledge-

ments in a sliding-window TCP transfer.[3] TCP ACKs are cumulative, so the loss of a single ACK causes no harm as long as another one gets through before the sending TCP fills its window.

7. Automatic Power Control

MACA lends itself well to automatic transmitter power control. To support this we need some extra hardware: a D/A converter that controls transmitter power level, and an A/D converter that gives received signal strengths. By including calibrated "S-meter" readings[4] in CTS packets, responders could help initiators to adjust their power levels accordingly.

Each RTS/CTS exchange updates the initiator's estimate of the power needed to reach a particular responder so that future packets (including the data packet in the current dialogue) can be sent with only the necessary power. Even RTS packets could be sent at reduced power, since their main purpose is to elicit a CTS from the responder. This reduces the probability of collision between RTS packets.

By changing the MACA rule to "inhibit a transmitter when a CTS packet is overheard" to "temporarily limit power output when a CTS packet is overheard," geographic reuse of the channel can be significantly improved. For example, if station X has recently sent traffic to station Y, it knows how much power is required to reach Y. If X overhears station Y responding with a CTS to a third station Z, then X need not remain completely silent for the required interval; it need only limit its transmitter power to, say, 20 dB [5] below the level needed to reach Y. During this time it would be free to transmit to any station that it could reach with that reduced power level, because its signal at Y would be overridden by Z's signal. (This is analogous to the people at the cocktail party continuing their conversations in whispers instead of stopping completely when Tom tells Bob to go ahead.)

The CTS packets, however, pose a problem. In addition to telling the initiator to send its data, the CTS must inhibit all potential interferers from transmitting. It may therefore need more power than that needed just to reach the initiator to ensure that everyone "gets the message." (A CTS packet might therefore be more like Tom shouting "Hey, everyone, shut up! I'm trying to hear Bob speak!" at the cocktail party mentioned earlier.)

All this shouting potentially limits the geographic channel reuse ability we've worked so hard to get. But all is not lost. A station responding to an RTS with a CTS can always expect data to follow. If it doesn't arrive within a reasonable period, or if a retransmitted RTS arrives instead, then either the CTS was stepped on, or the CTS wasn't heard widely enough to prevent the data transmission that follows from being stepped on. It should then respond to the next RTS from the same station (which will likely be a repeated attempt to send the same data) with a CTS at higher power. On the other hand, if a responder has had good luck in getting data in response to its CTS packets, it might try lowering the power it uses to transmit them in order to help limit channel loading. Of course, it would never lower its CTS power below the level it knows is necessary to reach the initiator.

In sum, MACA with power control automatically determines the exact amount of power required for each RTS and data transmission, and learns by experience (i.e., trial and error) the power required for CTS transmissions. It also appears to avoid the runaway power escalation that can occur when power control is done on a conventional CSMA channel when stations naively "turn up the wick" each time they fail to get through. About the only time power escalation seems possible in MACA is when an initiator's receiver fails so it is not able to hear CTS responses to its RTS packets no matter how much power the responder uses. This possibility should be handled by back-offs and/or retry limits in the dialogue code.

---

[3] The use of sliding windows in TCP might seem to contradict the advice I gave several years ago to always operate in stop-and-wait mode (**MAXFRAME** 1) on half duplex channels. However, that conclusion applied only to link level protocols; TCP is an end-to-end transport protocol. Sliding windows are usually appropriate in a transport protocol even when the individual hops in the network path are half duplex.

[4] Only one point in the S-meter scale really needs to be calibrated. This is the signal level just high enough to achieve an acceptable bit error rate. A more completely calibrated scale makes it easier for the transmitter to zero in on the correct power setting, but even a simple "too strong/too weak/OK" indication is enough for a transmitter to determine the correct power level by Newtonian iteration.

[5] This figure depends on the capture ratio of the modems in use.

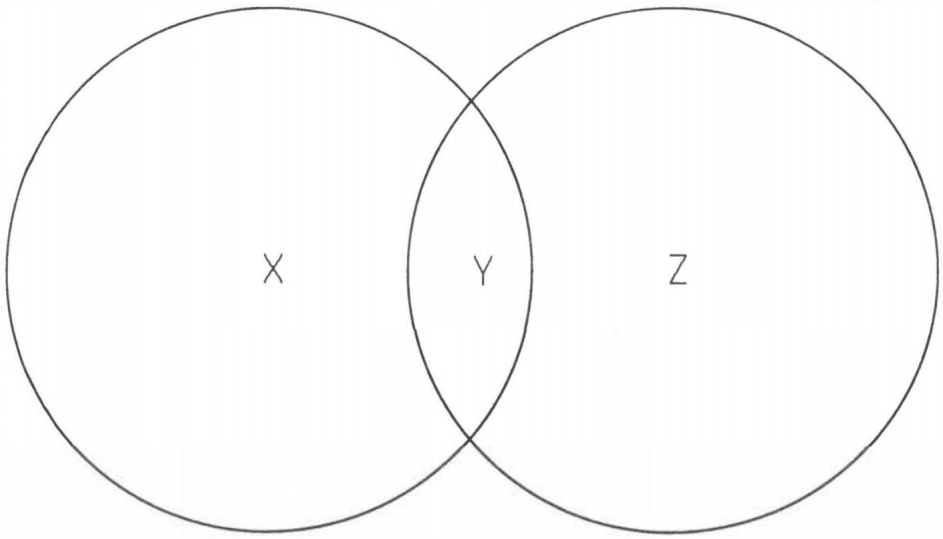
8. Applications for MACA

If MACA proves effective, it may *finally* make single-frequency amateur packet radio networks practical. Although it would still be preferable for fixed backbones to use separate, dedicated channels or point-to-point links whenever possible, the ability to create usable, ad-hoc, single frequency networks could be very useful in certain situations. These include user access channels (such as 145.01 MHz in many areas) and in temporary portable and mobile operations where it is often infeasible to coordinate a multi-frequency network in advance. This would be especially useful for emergency situations in remote areas without dedicated packet facilities.

An ideal emergency packet radio network would consist of identical stations operating on a common frequency (to maximize interchangeability) placed in arbitrary locations. These stations would automatically discover their neighbors and build routing and power control tables that maximize the total amount of traffic that can be carried in the coverage area. To do this, routing algorithms would use a different metric than usual. Instead of simply minimizing the number of hops needed to reach a given destination, the routing algorithm would instead minimize the *total transmitter energy* required by all the stations along a path to the destination. Because of the laws of RF propagation (doubling the range of a signal in free space requires four times as much transmitter power, and on the ground it may take much more), this approach would often *increase* the number of hops required to reach a given destination. However, overall network throughput would increase because the lower transmitter power levels would permit more simultaneous transmissions to occur in different parts of the network without interference. This would also minimize the power consumed at the stations, and this could be important when operating from batteries. The direct, minimum-hop path could still be provided as an option for special applications requiring minimum delay.

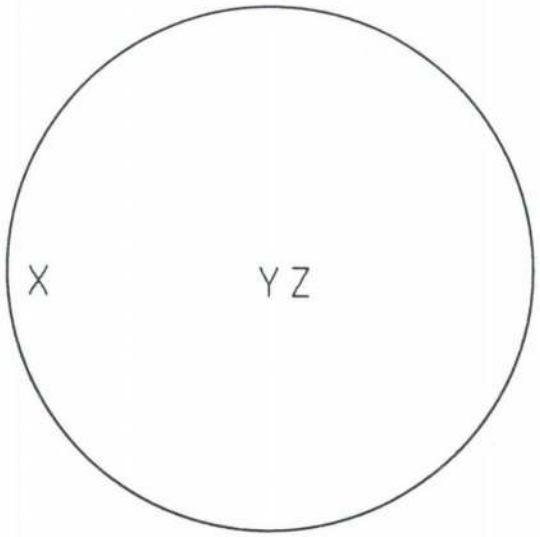
9. Conclusion and Open Questions

At the moment, MACA is just an idea. Much simulation and experimental work needs to be done to answer many questions about how well it will really work. Here are just some of the questions that can be asked. How much of the savings from avoided collisions in MACA is spent on RTS/CTS overhead given typical modem turnaround times and data packet sizes? How much better does power-controlled MACA perform than the basic MACA scheme? How about a partial implementation of power control, e.g., one that relies on trial-and-error instead of explicit S-meter feedback? How do the various forms of MACA behave as modem capture ratios change? How serious is the problem of interference from stations just below threshold? And how does MACA compare in overall spectral efficiency with other improved multiple access methods, such as conventional CSMA or CSMA/CD operation through full duplex repeaters? I invite anyone interested in pursuing these topics to contact me.



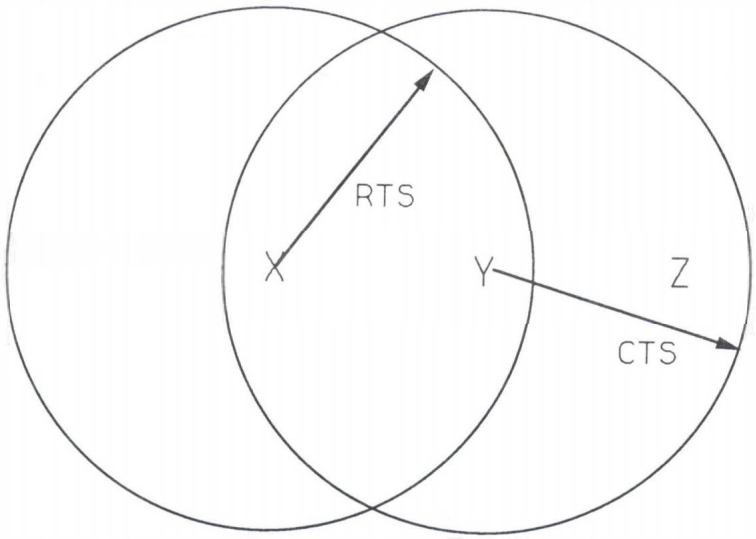
X and Z collide at Y

Fig 1. Hidden Terminal



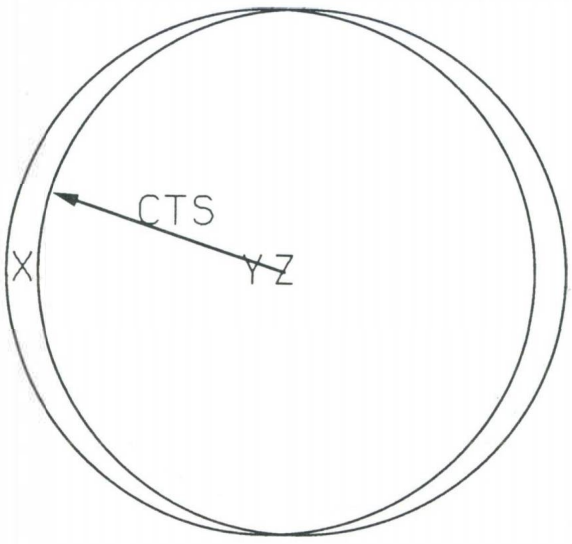
X defers unnecessarily to Y's transmissions to Z

Fig 2. Exposed Terminal



Y's CTS holds off Z

Fig 3. MACA with hidden terminal



X doesn't hear Z's CTS, so it doesn't need to defer to Y

Fig 4. MACA with exposed terminal

FORWARD ERROR CORRECTION FOR IMPERFECT DATA IN PACKET RADIO

W. Kinsner, VE4WK

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada R3T-2N2
E-mail: VE4WK@VE4BBS.MB.CAN.NA

Abstract

Many current protocols employ retransmission to ensure error-free data transfers. This is necessary for perfect data where a loss of a single bit is catastrophic. For imperfect data, such as digitized speech transmitted in real time in which a loss of one bit in a 1000 may be acceptable, retransmission is not possible and forward error correction should be used for error control. This paper presents a review of suitable codes for such error control, as well as several code implementations including a modified (8,4,4) Hamming code, (15,7) Bose-Chaudhury-Hocquenghem (BCH) code, and a concatenated code capable of correcting not only random errors but also burst errors. The code has an outer (7,4,3) Hamming code, an inner self-orthogonal 1/2 convolutional code, and a code word interlace matrix.

1. INTRODUCTION

There is a need for reliability in every communication system. A major problem in digital data communication systems is the introduction of errors due to a noisy channel. Unlike other parts of the system where transmission errors can be minimized or eliminated by careful design, the channel is not under control and transmission errors inevitably occur under noisy conditions. The existing error control scheme in amateur packet radio is retransmission whenever errors are **detected** in the received packets or when packets are lost. The detection is often based on either checksum words or cyclic redundancy code (CRC) words. However, retransmission is inefficient since it wastes both time and energy. When the communication medium is very noisy, retransmission may fail as every packet could contain errors. Moreover, retransmission is impractical for real-time applications and uni-directional communication. But above all, we have been using retransmission even though perfect data transmission is not required. Imperfect data such as non-critical text, digitized voice, and video may be acceptable with a small number or errors. Throughput of such systems could be increased considerably with a moderate error control by **correcting** (reconstructing) the bits in error at the receiving end, based on redundant bits inserted into the packet by the sender. This reconstruction scheme is called *forward error correction* (FEC), as opposed to the *automatic repeat request* (ARQ) or backward error correction (BEC) by retransmission.

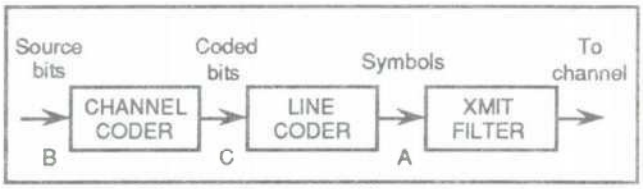


Fig. 1. Channel and line coders.

Coding refers to the translation between the *source bits* (user-provided message bits) and the transmitted *data symbols* (coded symbols) [Blah87, McEl87]. System performance can be improved by (i) *line coding* to control the power spectrum of the transmitted signal and (ii) *channel coding* to control errors [LeMe88]. The line coding changes the statistics of the data symbols to remove undesired correlations among the source bits so as to make them more random or uniformly active (through scrambling), while the channel coding introduces controlled correlation between data symbols through redundancy to detect and correct channel errors. Figure 1 shows a channel coder translating source bits into coded bits to achieve error detection, correction , or even prevention. The line coder further improves the probability of correct transmission. The decoding process may be *hard* (the inverse of the process of Fig. 1, with explicit decoding of all the symbols) or *soft* (direct decoding of the information bits from the received signal, without the detection of any intermediate symbols) [LeMe88].

Since this paper concentrates on the channel coding to handle random and burst errors simultaneously through simple block and convolutional codes, respectively, several pertinent definitions are required. Channel coding is concerned with two general classes of codes: block and convolutional. In the *block code* class, a sequence of source bits is segmented into blocks of k bits and translated into n code bits, with $n>k$ and $p=(n-k)$ redundant bits. The (n,k) code depends on the current block of k source bits only (*memoryless coder*). The block code is said to have code rate $r=k/n$.

On the other hand, a *convolutional coder* inserts redundant bits without segmenting the source stream into blocks. Instead, it processes the source stream either bit-by-bit or small groups of bits at a time. The code depends not only on the current input bits, but also on a finite number of past source bits (*coder with memory*). Convolutional codes produce results at least as good as the best block codes due to the availability of practical soft decoding techniques. Performance of an uncoded and coded system is measured by *coding gain* which is the difference in signal to noise ratio (SNR) required at the detector to achieve a fixed probability of either bit or symbol error.

By combining two or more simple codes, a good low-rate *concatenated code* can be constructed [MiLe85]. Here, the source stream is encoded with an (n,k) *outer code* and then further encoded as a sequence of (N,K) *inner code* blocks. The function of the outer code is to decode random or burst errors that have slipped through the inner decoder. The outer code may be made more effective if error bursts following the inner decoder are spread among consecutive outer code blocks by a process called *interleaving* or *interlacing* [Hayk88]. A simple line coder in the form of an interlace matrix can be used to assist in distributing such burst errors. Although the concatenated code is less powerful than a single-stage code with the same code rate and block length, the decoder is simpler due to the partitioning of the decoding stages.

2 BLOCK CODES

2.1 The Cross-Parity Code

One of the simplest error correcting codes is the cross-parity code based on horizontal and vertical parity checks, as shown in Fig. 2. Each word in the block has a horizontal parity bit added (to make the total number of 1s even). Vertical parity bits are computed from all the successive words and added to the columns. The parity bits can be generated by taking modulo-2 addition denoted by $\oplus$ (exclusive-OR gate). When a single error occurs in the block during its transmission (e.g., bit b_2 in the 3rd word is flipped to 0), then the parity bits for that row and column computed at the receiver site will be 1s rather than 0s, thus indicating the location of the bit in error which can be corrected by flipping. Correction of a single error is, therefore, possible without retransmission.

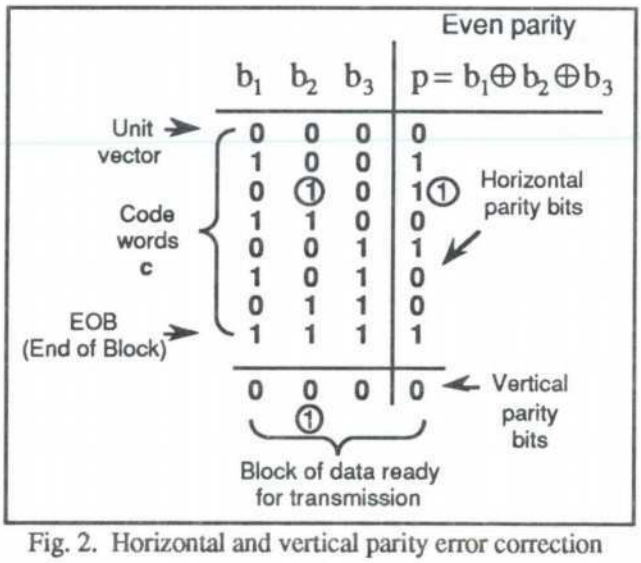


Fig. 2. Horizontal and vertical parity error correction scheme.

Notice that our numbering of bits in a word is from left-to-right, which is consistent with the majority of coding theory literature. This convention is related to matrix notation used in the early description of codes. This is contrast to the usual right-to-left number convention showing the weights of each digit in a number, as well as the modern literature that is based on polynomials, rather then matrices. The structure of the code can be described by total number of source bits

$$k = h \times v \tag{1}$$

where h and v are the numbers of rows and columns in the source block, respectively, the number of redundant parity bits

$$p = h+v+1 \tag{2}$$

and the total number of bits

$$n = k+p \tag{3}$$

giving the code rate of

$$r = k/n \tag{4}$$

For the code shown in Fig. 2, $k = 8\times3 = 24$, $p = 8+3+1 = 12$, $n = 24+12 = 36$, and $r = 24/36 = 2/3$.

The code is important not only from the educational point of view, but also due to its good code rate, no limit on the size of the source block, and the simultaneous single-error correction (SEC) *and* double-error detection (DED) capability, also called SEC-DED. Three errors may also be detected. What properties make the code suitable for correction? It is seen from Fig. 2 that any two code words differ by at least two bits. This is the most important observation determining the ability of a code to detect and correct a specific number of random errors. In other words, if the distance between two words is large enough, an error produces a **unique** word outside the transmitted set of code words, and the original code word can be reconstructed.

The number of places where two words (also called *row matrices* or *vectors*, denoted by bold characters) **u** and **v** differ is called the *Hamming distance* and is computed by

$$d(\mathbf{u},\mathbf{v}) = w(\mathbf{u}\oplus\mathbf{v}) \tag{5}$$

where $w(\bullet)$ is the *Hamming weight* defined as the number of nonzero elements in the resulting vector. For example, if we take the 3rd and 4th code words, then $d = w([0101] \oplus [1100]) = w([1001]) = 2$. The *minimum distance*, d_{min}, is the smallest distance between all the code words, and determines the number of errors that can be detected, t_d, and corrected, t_c, according to

$$d_{min} - 1 = t_d + t_c \qquad \text{for } t_c \le t_d \tag{6}$$

For example, if $d_{min} = 1$, no detection or correction is possible, as any error converts the code word into another valid code word. For $d_{min} = 2$, a single error can be detected but not corrected (the *h* parity bit). With $d_{min} = 3$, either a single error can be detected and corrected, or if the probability of a double error is high, then such a double error can be detected but not corrected. For $d_{min} = 4$, the SEC-DED scheme is possible. For $d_{min} = 5$, a double error can be detected and corrected. It is now seen why we need the *h* and *v* parity bits together; since the *h* bit leads to $d_{hmin} = 2$, it can detect an error in a row only, and the *v* bit ($d_{vmin} = 2$) is needed to locate the column where the bit in question is. Note that the cross-parity code is an example of a class of codes called *product codes* in which the total distance is $d_h d_v$ and the individual codes are not limited to the parity check only.

The block code of Fig. 2 has also another useful *linear* property because the modulo-2 sum of code words produces another code word. For example, the 3rd and 4th code words result in the 2nd code word ([0101] ⊕ [1100]) = [1001]). This property makes encoding and decoding easy by using linear (X-OR) operations rather than random table look-up procedures.

The encoding and decoding may be further simplified because the code is also *systematic* (all the parity bits are separated from the source bits). Still another useful property of the code is its *cyclic* form; i.e., when a code word is rotated (shifted cyclically), the resulting word is also a code word. For example, by rotating the 3rd code word on place to the right, the 6th code word results ([0101] → [1010]).

The code word generation process can be expressed conveniently using matrix notation. Notice that a row code vector **c** is a concatenation of the row source vector **b** and the parity bit p, which generalizes to

$$\mathbf{c} = \mathbf{b}\mathbf{G} \tag{7}$$

where **G** is the *generator matrix* given by

$$\mathbf{G} = [\mathbf{I}_k, \mathbf{P}] \tag{8}$$

and $\mathbf{I}_k$ is the identity matrix of dimension $k\times k$ and **P** is the single column parity matrix. For the example of Fig. 2, the 2nd code word is generated by

$$[0011] = [001]\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{9}$$

2.2 Hamming Codes

2.2.1 The Nonsystematic (7,4,3) Hamming Code

In the 1950s, Hamming introduced a class of linear, cyclic, systematic and perfect correcting codes with distance of 3 and 4. A *perfect* code is one in which every bit in the code word may be corrected. The simple (7,3) Hamming code has $n = 7$ total bits, including $k = 3$ source bits and $p = 4$ parity bits (heavy overhead). It can be generated using LFSRs and correction can be done using majority logic. The (7,4) Hamming code improves the code rate and allows operation on 4-bit nibbles. Correction of a single error is done using a parallel syndrome vector generator which is the address of the bit in error. The simple syndrome vector is obtained by a proper coverage of the source bits by the parity bits, as shown in Fig. 3.

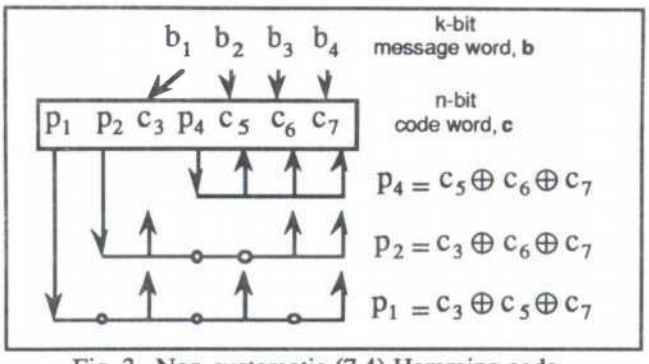


Fig. 3. Non-systematic (7,4) Hamming code.

For illustrative purposes, the source bits are interspersed with the parity bits so that the parity bits could occupy positions represented by the power of 2 (i = 2^0=1; = 2^1=2; = 2^2=4;...). As shown by the arrows in Fig. 3, the index i of each p_i shows: (i) the group of bits to be checked for parity and (ii) the separation between the groups. This provides a binary coverage of the bits, with the most significant bit affecting all the parity bits, and the other bits affecting two parity bits (e.g., c_5 affects p_1 and p_4. For example, if

$$\mathbf{b} = [b_1 b_2 b_3 b_4] = [c_3 c_5 c_6 c_7] = [0101] \quad (10)$$

then the parity bits are computed as

$$p_1 = c_3 \oplus c_5 \oplus c_7 = 0 \oplus 1 \oplus 1 = 0$$
$$p_2 = c_3 \oplus c_6 \oplus c_7 = 0 \oplus 0 \oplus 1 = 1$$
$$p_4 = c_5 \oplus c_6 \oplus c_7 = 1 \oplus 0 \oplus 1 = 0 \quad (11)$$

and the code word is

$$\mathbf{c} = [p_1 p_2 c_3 p_4 c_5 c_6 c_7] = [0100101] \quad (12)$$

If we assume that **c** is transmitted as is (without any line coding) and that an error occurs in position c_5, then the received code word is **c*** = [0100001]. The syndrome vector, **s**, is calculated as

$$s_1 = p_1 \oplus c_3 \oplus c_5 \oplus c_7 = 0 \oplus 0 \oplus \mathbf{0} \oplus 1 = 1$$
$$s_2 = p_2 \oplus c_3 \oplus c_6 \oplus c_7 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$$
$$s_4 = p_4 \oplus c_5 \oplus c_6 \oplus c_7 = 0 \oplus \mathbf{0} \oplus 0 \oplus 1 = 1 \quad (13)$$

It is seen that since $s = [s_4 s_2 s_1] = [101]$ is the address of the location with error, an ordinary parallel binary decoder can correct the error.

The (7,4) code concept can be extended to more bits (n,k) with

$$p = n - k, \quad n = 2^p - 1, \quad k = n - p \quad (14)$$

For $p = 4$, $n = 16\text{-}1 = 15$ and $k = 15\text{-}4 = 11$ we have a (15,11) code with 11 source bits and a much improved code rate of $r = 11/15$.

The descriptive formulation of the (7,4) code can be presented in a more formal manner by using the following *$p \times n$ parity-check H matrix*

$$\mathbf{H} = [\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3\ \mathbf{h}_4\ \mathbf{h}_5\ \mathbf{h}_6\ \mathbf{h}_7] = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (15)$$
$$[p_1\ p_2\ c_3\ p_4\ c_5\ c_6\ c_7]$$

Notice the binary-coded structure of the matrix from left to right. This matrix can can be used to construct any other distance-3 code. (For the distance to be at least 3, no two columns in H should be the same, or any three columns should add to 0.)

2.2.2 The Systematic (7,4,3) Hamming Code

The above analysis concentrated on a non-systematic form of the linear code which require parallel circuits for its generation and decoding. In practice, serial circuits (LFSR) are preferred since the bits are transmitted serially. To find an appropriate LFSR for the (7,4) code, the code must be cyclic and systematic. This can be achieved by changing the **H** matrix to a *$k \times n$ generator matrix* from which a *generator polynomial* could be derived. First, the matrix **H** can be changed from a non-systematic form to a systematic form by rearranging the columns for the most convenient encoding of decoding (this change preserves the SEC capability due to the uniqueness of the columns)

$$\mathbf{H} = [\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3\ \mathbf{h}_4\ \mathbf{h}_5\ \mathbf{h}_6\ \mathbf{h}_7] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (16)$$
$$[c_5\ c_7\ c_6\ c_3\ p_4\ p_2\ p_1]$$

$$\mathbf{H} = [\mathbf{P}^T \mid \mathbf{I}_p] \quad (17)$$

where **P** is the $p \times k$ coefficient matrix that defines how the parity bits are related to the message bits, T denotes transpose, and $\mathbf{I}_p$ is the $p \times p$ identity matrix. This systematic form is also called *reduced echelon form* [PeWe72] and can be obtained formally by row reduction into the canonical form [LeMe88]. It can be shown [RaFu89, MiLe85, LeMe88, Hayk88, ClCa81, Kuo81, Rode82, MiAh88] that the generator matrix is related to the parity-check matrix through

$$\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}] \quad (18)$$

where I is the $k \times k$ identity matrix. From Eq. 16, the (7,4) code has the following generator matrix

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \\ \mathbf{g}_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{19}$$

and can be generated by taking

$$\mathbf{c} = \mathbf{bG} \tag{20}$$

Note that the generator vector $\mathbf{g}_4$ in Eq. 19 is important because it specifies the generator matrix G completely. For example, the vector $\mathbf{g}_3$ is obtained by shifting $\mathbf{g}_4$ one place to the left. Next, vector $\mathbf{g}_2$ is obtained by shifting $\mathbf{g}_3$ to the left and adding (modulo-2) $\mathbf{g}_4$ to it. Finally, $\mathbf{g}_1$ is obtained by shifting $\mathbf{g}_2$ left and adding $\mathbf{g}_4$. Therefore, all the code words are generated by cyclic shifts of $\mathbf{g}_4$.

The generator vector $\mathbf{g}_4$

$$\mathbf{g}_4 = [\,0\ 0\ 0\ 1\ 0\ 1\ 1\,] \tag{21}$$

is associated with a generator polynomial of the following general form

$$\mathbf{g}(x) = x^p + \sum_{i=1}^{p-1} g_i x^i + 1, \quad g_i = \{0,1\} \tag{22}$$

where p is the number of redundant bits. For the (7,4) code, p=3 and g(x) becomes

$$\mathbf{g}(x) = x^3 + x + 1 \tag{23}$$

More formally, the generator polynomial is found be factoring (x^n-1) into irreducible polynomials and selecting a primitive one of degree $p=(n-k)$.

We can now rewrite Eq. 19 in the polynomial form as

$$\mathbf{c}(x) = \mathbf{b}(x)\mathbf{g}(x) \tag{24}$$

and the code is obtained by multiplying the message word with the generator polynomial using either parallel circuits or serial LFSRs. The problem here is that the multiplication almost always produces a non-systematic code. However, the message bits are not altered if they are just shifted to the left by as many places as the degree of the generator polynomial g(x). This is equivalent to the multiplication of $\mathbf{b}(x)$ by x^p. But to maintain $\mathbf{c}(x)$ unchanged, the right-hand-side of Eq. 24 must be adjusted by the remainder polynomial

$$\boxed{\mathbf{c}(x) = x^p\mathbf{b}(x) + \text{rem}\left(\frac{x^p\mathbf{b}(x)}{\mathbf{g}(x)}\right)} \tag{25}$$

where rem(•) denotes the remainder. The above expression constitutes the encoding algorithm for LFSRs: the message word $\mathbf{b}(x)$ is shifted to the left by p bits, and the p parity bits represented by the rem(•) are appended to the message word.

2.2.3 A (7,4,3) Hamming Encoder

The most convenient circuit to compute the rem(•) is a LFSR whose structure is represented by the selected generator polynomial g(x). It is shown in Fig. 4.

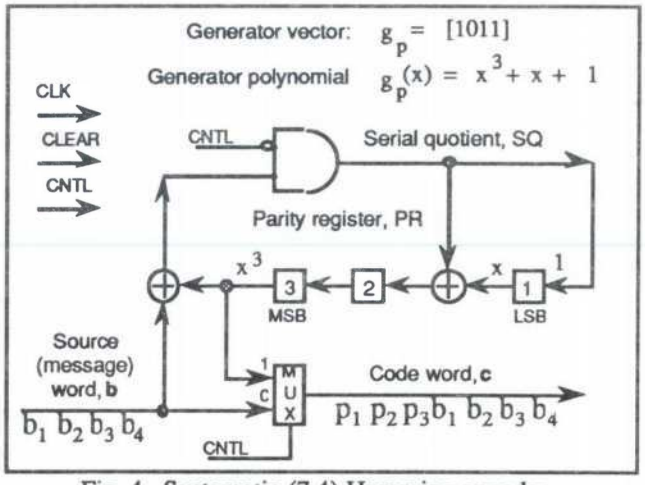


Fig. 4. Systematic (7,4) Hamming encoder.

The LFSR has p=3 stages (flip flops, FFs) in its *parity register*, PR, whose contents are first set to **0** at the beginning of each new message word, and then shifted by a clock. The stages are separated by modulo-2 adders at places determined by the $\mathbf{g}(x)$. The LFSR multiplies $\mathbf{b}(x)$ by x^p at its entry, and subtracts g(x) from its current contents whenever the *serial quotient* (SQ) is 1. The SQ is calculated from

$$SQ^{(t)} = FF_p \oplus b_{(k-t)}, \quad t=0,1,\cdots,k-1 \tag{26}$$

where t denotes clock period, FF_p is the most-significant bit (MSB) of the FF, and $b_{(k-t)}$ are the successive message bits, starting from the MSB.

Table 1. Remainder generation in a LFSR.

CYLCE	CNTL	SQ	DATA IN	REGISTER MSB	REGISTER LSB	OUT PUT
CLEAR	0	1	**1**	⊕0 0	⊕ 0	**1**
Shift 1	0	0	**0**	⊕0 1	⊕ 1	**0**
2	0	0	**1**	⊕1 1	⊕ 0	**1**
3	0	1	**0**	⊕1 0	⊕ 0	**0**
4	1	0	x	0 1	⊕ 1	**0**
5	1	0	x	1 1	0	**1**
6	1	0	x	1 0	0	**1**

Let us rewrite Eq. 9 in reverse sequence, consistent with polynomial notation

$$\mathbf{b} = [b_4 b_3 b_2 b_1] = [1010] \tag{27}$$

and trace the events in the encoder, using Fig. 4 and Table 1. Following a CLEAR, the feedback path is closed by the application of CNTL=0, the multiplexer MUX connects the source input, **b**, to the coder output, and the SQ is generated as $SQ^{(0)} = FF_3 \oplus b_4 = 0 \oplus 1 = 1$. After the first shift, $FF_1 = 1$, $FF_2 = FF_1 \oplus SQ^{(0)} = 0 \oplus 1 = 1$, $FF_3 = 0$ from FF_2, and $SQ^{(1)} = FF_3 \oplus b_3 = 0 \oplus 0 = 0$. This process continues for three more clock periods, and the final remainder $\mathbf{p} = [p_3\ p_2\ p_1] = [011]$ is appended to the message word by turning the feedback path off, switching the MUX to the PR, and applying p=3 extra clock periods to produce the correct code word $\mathbf{c}$ = [1010011].

The above operation can be confirmed using polynomial manipulations. The message vector of Eq. 27 has the following corresponding polynomial

$$\mathbf{b}(x) = x^3 + x \tag{28}$$

and the remainder

$$\mathbf{p}(x) = \text{rem}\left[\frac{x^3\,(x^3 + x)}{x^3 + x + 1}\right] \tag{29}$$

is computed using Euclidean long division as follows.

$$\begin{array}{r} x^3 + \quad 1 \\ x^3 + x + 1 \overline{)\,x^6 + x^4} \\ \underline{x^6 + x^4 + x^3} \\ x^3 \\ \underline{x^3 + x + 1} \\ \text{Remainder } p(x) = x + 1 \end{array} \tag{30}$$

By changing the remainder into its matrix form $\mathbf{p} = [011]$ and concatenating it with **b**, **c** is found as

$$\mathbf{c} = \mathbf{b} \mid \mathbf{p} = [1010011] \tag{31}$$

Similarly, by using Eq. 25, we find

$$\begin{aligned} \mathbf{c}(x) &= x^3\,(x^3 + x) + (x + 1) \\ &= x^6 + x^4 + x + 1 \end{aligned} \tag{32}$$

or $$\mathbf{c} = [1010011] \tag{33}$$

2.2.4 A (7,4,3) Hamming Decoder

Since syndrome generation can be considered as the inverse of code word generation, it can use the same LFSR multiplier/divider. The syndrome vector **s** is computed according to Eq. 25

$$\begin{aligned} \mathbf{s}(x^p \mathbf{c}^*(x)) &= x^p \mathbf{c}^*(x) \bmod (\mathbf{g}(x)) \\ &= \text{rem}\left(\frac{x^p \mathbf{c}^*(x)}{\mathbf{g}(x)}\right) \end{aligned} \tag{34}$$

If there is no error in $\mathbf{c}^*(x)$, then **s**=0. If an error $\mathbf{e}(x)$ occurs, then the received signal is linear combination of **c** and **e**

$$\mathbf{c}^*(x) = \mathbf{c}(x) + \mathbf{e}(x) \tag{36}$$

and the syndrome is due to the the error only

$$\begin{aligned} \mathbf{s}(\mathbf{c}^*(x)) &= \mathbf{s}(\mathbf{c}(x)) + \mathbf{s}(\mathbf{e}(x)) \\ &= \mathbf{c}(x) \bmod (\mathbf{g}(x)) + \mathbf{e}(x) \bmod (\mathbf{g}(x)) \\ &= 0 + \mathbf{e}(x) \bmod (\mathbf{g}(x)) \\ &= \mathbf{s}(\mathbf{e}(x)) \end{aligned} \tag{37}$$

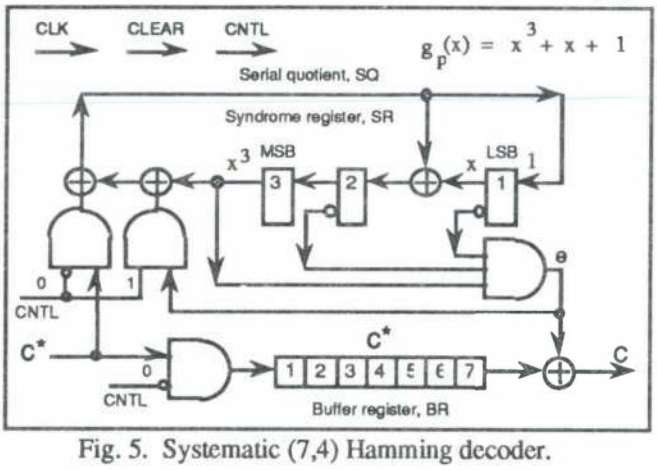


Fig. 5. Systematic (7,4) Hamming decoder.

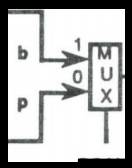
Figure 5 shows how the received word $\mathbf{c}^*(x)$ is fed to the syndrome generator and a buffer register, BR, while CNTL=0. After n=7 shifts, the syndrome is in the syndrome register (SR) and a copy of $\mathbf{c}^*$ is in BR. The control signal now becomes CNTL=1 to break the path of input $\mathbf{c}^*$ into the LFSR and BR, as well as to open a new feedback loop for the error correcting signal *e*. The error is corrected after another n clock periods.

Table 2. Syndrome vectors for systematic (7,4) code.

e(x)	s(e(x))	s(x³ e(x))
1	1	x+1
x	x	x²+x
x²	x²	x²+x+1
x³	x+1	x²+1
x⁴	x²+x	1
x⁵	x²+x+1	x
x⁶	x²+1	x²

$x^6)_2$

$x^3)_2$



$.. b_2 b_1$

of a

3.2 Self Orthogonal Codes

A subclass of convolutional codes is the *canonical self-orthogonal code* (CSOC) [ClCa81] in which syndrome symbols can be taken directly as estimates of the source stream, using majority logic. An optimal 1/2 CSOC of constraint length $L=(6+1)\times2 = 14$ was designed for digital radio at AT&T [Mart85]. The code is optimal in the sense of the shortest registers used. An encoder for the code is shown in Fig. 7.

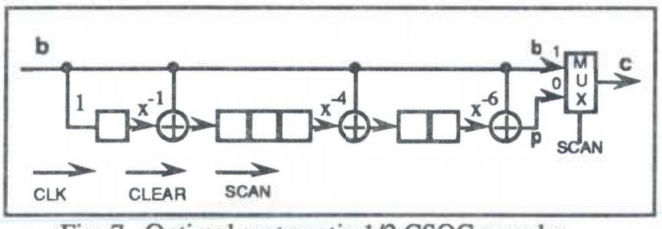


Fig. 7. Optimal systematic 1/2 CSOC encoder.

A CSOC decoder is shown in Fig. 8. The received bits stream is split into message bits **b** and parity bits **p**. The **b** bits are shifted into a buffer register and into a CSOC parity encoder for regeneration of the parity bits **p****; if the bit from the parity encoder, p^{**}_i, and the currently received parity bit p^*_i are identical, no error has occurred at that bit position. The syndrome bits are then delayed through a syndrome register, SR, and five of them are passed to the majority circuit which produces a 1 whenever at least three inputs are 1; if b^*_{i-6} is incorrect and e=1, b^*_{i-6} is corrected. If b^*_{i-6} is correct and not more than two errors are allowed in the previous 13 b* bits and 4 p* bits, then b^*_{i-6} will not be corrected falsely. The bit error rate of the definite decoder (without feedback), BER_{DD}, for triple errors at a received BER = 10^{-3} is $BER_{DD} \approx 3.5\times10^{-7}$ [Mart85]. If the e signal is fed back into the SR, then a corresponding adjustment is made to the other syndromes, leading to an improved BER_{FD}. This encoder and decoder was used in the experiments described next.

4. EXPERIMENTS

4.1 Block Codes

A systematic (8,4,4) Hamming code encoder and decoder was built to test the transmission of ASCII characters and speech compressed using linear predictive coding (LPC-10) [Swan87] over noisy channel [ChXu89]. The Hamming circuitry described in the previous section was built using TTL components, controlled by a 68000-based laboratory system with software written in the 68000 assembly language. The important finding from this work was that a paragraph of English text transmitted with FEC over a channel with double and mixed errors was read by a group of testers without difficulties. Experiments with LPC speech indicate that the above scheme improves the quality of the bits transmitted (23 to 50 %), but is not sufficient due to the high sensitivity of the speech parameters to errors.

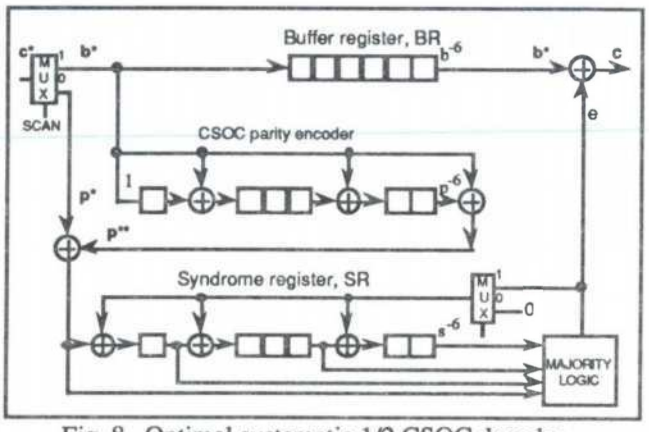


Fig. 8. Optimal systematic 1/2 CSOC decoder.

A (15,7) BCH encoder and decoder was also developed to study their complexity [LaCh90]. The code can correct double random errors and correct burst errors of length 4. A TTL implementation of the code was controlled by an IBM PC through an interface and software written in Microsoft Quick C and 8086 assembly language.

4.2 The Concatenated/Interlaced Code

The concatenated code used in our latest experiment [KwNa90] consisted of the (7,4) Hamming code, serving as an outer code, and the 1/2 CSOC convolutional code serving as an inner code – both described in the previous section. By using an interlace matrix capable of holding 8 code words, burst errors were distributed over a range of bit positions so that the outer simple Hamming code could handle them better. A controller based on the 6802 microprocessor was used to handle interfacing. The encoder, decoder and interlacing circuits were realized in TTL. Most of the software was written in the 6800 assembly language. Performance of the code was measured using probability of character error (PCE) rather

adding new features to this code.

Jeffrey Comstock, NRØD, is writing a full NNTP server and client. The local user can decide whether the client will offer articles for forwarding, or if it should request articles.
This client requests the article in full, as well. The articles are not converted into mail format, so they cannot be accessed from the NOS BBS. Instead it is possible to read and write articles using various more advanced news readers that are external to NOS, such as WAFFLE and the Japanese Terakoya system.

The server is not finished at the time of writing, but an early alpha release is available for testing.

Conclusion

There are at least two NNTP clients and one server implemented for the KA9Q Internet Package. Both clients request whole articles at once, instead of just the header lines. In an efficient implementation of NNTP it should be optional to transfer only the headers first, and then let the local user decide if the rest of the article is wanted. But this will require changes to the existing BBS in NOS, or porting of second source news software to the NOS environment.

References

Horton 87 M. Horton, "Standard for Interchange of USENET Messages", *ARPA RFC-1036*, December 1987.

Kantor 86 B. Kantor, P. Lapsley, "Network News Transfer Protocol, A Proposed Standard for the Streams-Based Transmission of News", *ARPA RFC-977*, February 1986.

PACKET RADIO WITH RUDAK II ON THE RUSSIAN RADIO-M1 MISSION

Hanspeter Kuhlen, DK1YQ
AMSAT-DL RUDAK Group
Finkenstrasse 11; D-8011 Aschheim nr. Munich
Federal Republic of Germany

INTRODUCTION

Let´s face it ! RUDAK, the digital regenerative transponder on-board OSCAR 13 was a grievous loss. No need to say that this was a very big disappointment not only for the RUDAK group of AMSAT-DL but also for many satellite packeteers in the world.

Early 1989 the AMSAT-U-ORBITA group, represented by Leonid Labutin, UA3CR, of the USSR offered AMSAT-DL another opportunity to fly a second generation of the RUDAK experiment commencing an up to that point unprecedented Russian/West German cooperation in amateur radio satellite activities. It was during the annual AMSAT-UK colloquium in Surrey 1989 when representatives of both organizations met to sign a memorandum of understanding. Subject of this MoU was to design, manufacture and launch a joint transponder mission on-board the Russian scientific satellite GEOS. The satellite is to be launched by a Russian launcher in fall 1990.

This paper will report on the mission objectives and the sofar achieved results. The transponder has been named RADIO-M1 by AMSAT-U-ORBITA. A "one" because it is our first joint project and "M" because the involved AMSAT groups are located in **M**olodechno near **M**insk (ORBITA), **M**oscow and **M**unich (RUDAK). Once the satellite is in orbit it will probably become RS 14 (Radio Sputnik).

The main part of the RADIO-M1 transponder is dedicated to packet radio operations as it has become normal practise also by the other packet radio satellites such as FUJI OSCAR and MICROSATs. The RUDAK operational software is ready for a long time. It has been field tested with the engineering model of the transponder from top of the watertower in Ismaning near Munich in South Germany over several years and proven to be operational.

The other part serves as a high-speed, multi purpose computer facility good for a number of different communication experiments including digital signal processing (DSP).

SYSTEM OBJECTIVES

The RUDAK I system was ideally suited for the particular advantages of highly inclined elliptical orbits (HEO) of the AMSAT Phase 3 satellites. One of the main advantages of these

HEO orbits is that, at least in the Northern Hemisphere, the satellite is visible for many hours daily. Therefore, RUDAK I has primarily been designed to feature "real-time" packet radio communications (digipeating) and, in addition to that, provide minimum mailbox functions in the so-called "ROBOT" mode.

The ROBOT mode allows direct connects to the RUDAK processor by using "RUDAK" as a callsign with the extender (SSID 0-15) standing for a few standard messages. Those messages include e.g. latest orbit information (Keplerians), transponder schedules, selected telemetry parameters, uplink statistics, mheard list and so on. These messages are transmitted in response to a connect request under full control of the AX.25 protocol, i.e. error corrected. It will download the information to the connected user terminal followed by an automatic disconnect initiated by the satellite as soon as the message transfer is completed.

Different from the Phase 3 satellites on elliptical orbits will GEOS travel along a circular earth orbit at an altitude of 1000 km almost over the poles (83° inclination). Consequently we have similar conditions in terms of visibility, doppler and so forth as we already know from the FUJI satellites. The RUDAK system itself demonstrated excellent performance for more than two years from top of the watertower in Ismaning near Munich, FR Germany.

When the RUDAK group had their first system meeting on RUDAK II in August 1989 (!) it became very soon clear that due to the extremely short project time of not more than six months it would be better to stick with the AO13 design and simply to repeat it. RUDAK I can shortly be described by a general purpose computer with special protocol interface devices (Z80 SIO) to run the AX.25 protocol. At least, to be more precise, this would have been the most logical conclusion. However, in the course of further discussions on the system performance and the actual state-of-the-art in processor technology it became very quickly obvious, that a simple re-built of the equipment appeared not very attractive to us.

First, because RADIO-M1 will be flying at a low altitude of 1000km. It has been said before that from this orbit you get many visibility periods a day but only at relatively short intervalls of about 10 to 20 minutes. Therefore, the digipeat mode, which would be very desirable from an elliptical orbit is much less interesting from an low earth orbit (LEO). In LEO preferably operating modes such as file transfer (ROBOT/ Mailbox) and store-and-forward are much better suited.The second and probably more important reason for not only re-building the former RUDAK was the missing challenge.

Eventually, after thorough investigation of several options we concluded in the following mission objectives

1 demonstrate full RUDAK 1 operation
2 provide a user friendly communications transponder, compatible with present amateur packet radio satellites (FUJI, Microsats, UoSat)
3 introduce additional experimental modes with higher bitrates, different modulation schemes etc. to provide a flying testbed for demonstration of new communication methods and software (multiple access, store-and-forward, signal processing etc.)
4 to fly advanced technology (processors and VLSI circuits) to study their behaviour under real application conditions gaining experience for future satellite projects (Phase 3D etc.)

Admittedly, the objectives became quite ambitious in the end and the resulting payload architecture grew slightly more complex.

PAYLOAD ARCHITECTURE

Figure 1 shows a blockdiagram illustrating the key features of the RADIO-M1 system. The system consists mainly of two sections, an RF receiver/transmitter and a digital processor section. Both are interconnected by a baseband signal switch and hard-command decoder. The hard command decoder processes the PN sequences to enable a secure command access to the transponder without use of on-board computer resources.

In OSCAR 13 this service was provided by the main computer (Integrated Housekeeping Unit). The RF section has been designed and jointly built by AMSAT-U-ORBITA and AMSAT-DL. It comprises four independent receiver channels (uplinks) in the 70cm band and a single multi-mode transmitter (downlink) on 2m. Reference is made to Table 1 for exact frequency information. The RF front-end, two linear downconverter and the power amplifier were built by AMSAT-U.

The digital section comprises two independent computers of which one is the exact replica of the OSCAR 13 processor and the second is a very fast digital signal processing (DSP) computer with a so-called reduced instruction set (RISC) architecture.

RECEIVER AND TRANSMITTER SECTION

Digital uplink signals received on 70cm are delivered on a 10.7 MHz intermediate frequency as a 400 kHz wide signal. Selectivity is provided in four receivers respectively each operating on a different frequency in the IF range. Each of the four receivers has its own attraction

providing special flavour to the RUDAK experiment.

The first receiver (RX-1) is equipped with an FM discriminator followed by a bit clock regeneration circuit to achieve a clean (regenerated) data and clock signal at a bitrate of 1200 bit/s. As shown in Figure 1, the FM demodulated baseband signal is further connected via an A/D converter to the digital signal processor.

This receive path will mainly be used for normal packet uplinking since bitrate, modulation scheme and coding is fully compatible with FUJI OSCARs and the other AMSAT packet radio satellites. Except for the mode U configuration, i.e. 70cm up and 2m down, which is reversed to FUJI operation, the same ground user terminal equipment can therefore be utilized for RADIO-M1 operation as well. During certain periods of time, which will be announced through appropriate bulletins, experiments may utilize this link for other purpose, e.g. voice processing from an FM voice uplink etc..

The second receiver (RX-2) is indeed mostly a re-built of the former AO13 receiver. It demodulates 2400 bit/s in digital phase modulation (BPSK). The receiver scans around its nominal center frequency for ± 10 kHz at a scan rate of 120 msec. Therefore, doppler shift and other contributions to uplink frequency ambiguity are to a great extend compensated by the satellite receiver. So, nominally the user may set his transmitter to the nominal center frequency provided in Table 1 and the satellite receiver will quite probaly track your uplink signal. Both receiver, RX-1 and RX-2, are considered the main receivers for normal packet operation. They will normally be connected to the R1 processor.

The third receiver (RX-3) is connected to two demodulators operating at bitrates of 4800 and 9600 bit/s respectively. This system operates in both bitrates with a new type of modulation, named Rectangular Spectrum Modulation (RSM). This highly efficient modulation scheme was invented some years ago by Dr. Karl Meinzer, DJ4ZC. RSM is applied through optimum filtering and preshaping of the transmitted binary signals and a matched filter on the receive side. RSM provides optimum bandwidth performance under minimum inter symbol interference. Further information on RSM is available on request from AMSAT-DL. RX-3 is also a frequency scanning type of receiver scanning ± 10kHz around nominal frequency.

Finally, last but not least, the most sophisticated part of the experiment is the receiver RX-4 consisting of only minimum hardware to provide baseband signals as an analog inphase (I) and quadrature phase (Q) signal to the RISC computer. With this configuration the actual demodulation and decoding process is left with the DSP, in other words with software. The characteristic and performance of the receive path is only limited by its RF-bandwidth of 30 kHz maximum and the maximum speed and capability of the application software. This link should allow bitrates of up to the order of 20 .. 25 kbit/s at any vector modulation be it BPSK,

QPSK, 16-QAM or what have you. It is probably redundant to emphasize that this part of the experiment is an Eldorado for communication and software enthusiasts.

The transmitter is an entirely new design. It has multi mode operation capabilities. It can operate at any bitrate up to 9600 bit/s since the actual bitrate, its coding and even the type of modulation is under control of the processor, thus software. Table 1 gives an impression on the various modes of operation. The nominal output power on 2m is 3W.

PROCESSOR SECTION

The processor section comprises two independent computers. The first is actually a clone of the AO13 RUDAK processor. The CPU is a 65SC02 CMOS 8-bit processor with 56 kByte of SRAM. The operating system is IPS, which is a derivate of FORTH ® but with very powerful add-ons, e.g. enabling multi processing. IPS has proved its viability in OSCARs 10 and 13 and has also been adopted for the RUDAK system.

The computer is booted from a ROM based loader. The ROM device is a so-called fusible link PROM of 2 kByte capacity. It contains actually a minimum operating system, the ROM operating system (ROS). ROS provides a number of useful utilities beside initializing the computer and preparing it for uploading of the operational software. The RUDAK system on GEOS will permanently be connected to the main power bus of the spacecraft.

It is for the time beeing not yet shure, if the system will be launched with its operational software already resident or not. Presumably it will be not resident. In this case the ROS system will transmit telemetry information at a 400 bit/s bitrate in a format compatible with the OSCAR 13 telemetry. But except for certain engineering tests later, this mode is only used for a short time during initial performance checks.

Two input channel into the computer can be used for communication through a sophisticated SIO device (Z80 SIO). This device supports significantly the AX.25 protocol calculations unloading the computer from excessive number crunching and counting, thus leaving more computer power for the higher level tasks. The two input channels are configured in a way that one channel is permanently set to 1200 bit/s operation while the second can be connected to any of the receivers RX-1 to 3 and operated at variable bitrates. The switching is performed by the baseband switch under software or hard telecommand control.

A 1 MByte RAM has been added as a universal mass storage device. It is addressed in a similar manner as a floppy disk drive in a normal computer system, i.e.through an I/O device. Therefore it is called RAM-Disk. The RAM-Disk will mainly be used for the mailbox but it may also store experimental software which is to be invoked by either of the two computer for special communication tests.

A bi-directional 8-Bit interface interconnects both computer and the RAM-Disk. Through this interface one computer can support the operation of the other. For instance, one computer can take care of all AX.25 protocol aspects while the second searches the mailbox for special files at a very fast speed. This double-computer architecture is also advantageous if a problem with one processor occurs. The redundancy will in this case be used to maintain normal packet operation.

The real challenge of the RUDAK II system is no doubt in the second computer. It is basically a reduced instruction set computer which runs at the very high processing speed of about 10 MHz (CPU: RTX 2000) providing 100 ns processor cycles. The CPU may be operated with or without wait states in error correcting or non-correcting mode. Error detection and correction of single failures is incorporated by means of a special VLSI-device (EDAC: 39C60). This configuration leaves greatest flexibility in the operation of the equipment under outer space conditions which are mainly characterized by various types of cosmic radiations. Fast memory with 35 nsec typical access time (we measured actually 15 nsec with the real devices!) at a quantity of 128 kByte (plus 64 kByte for error correction) opens a wide field of signal processing possibilites, in other words application software.

In this context it is interesting to note, that the main reason for selecting the RTX 2000 CPU was its internal FORTH® oriented hardware structure. This structure actually comprises IPS definitions in hardware. That, together with the high speed operation of the hardware leaves plenty of room for ambitious expectations.

Different from the R1 processor, the RTX has a direct memory access (DMA) logic to allow secure uploading of loader software. This hopefully will avoid a similar lock-up condition which occured on OSCAR 13 RUDAK with the initialization process.

BASEBAND SWITCH

The baseband switch serves two purposes. First, it flexibly can interconnect all receiver outputs with the communications processor R1 while the RTX computer is permanently connected to all receivers. In return, the transmit signal generated by either of the two processors can be switched onto the downlink. For ranging or checking of the uplinks any of the receivers can alternatively be connected to the downlink.

The second major task of this unit is the generation of a complex PN binary sequence to decode commands from ground command stations. To this end two pairs of receivers, actually bitrates, can be routed to the processors alternatively. Either of the receive channel can be used to upload software or commands to perform engineering tests requiring access to the operating system.

POWER SECTION

The amateur payload RADIO-M1 of the host satellite GEOS will remain permanently connected to the main power supply bus. Four independent DC/DC-converter are provided to power up the various functional groups independently. One for the RF section, another two for each of the processors and finally one for the RAM-Disk. This provides a relatively high degree of independence from any failure condition.

RADIO-M1/ RUDAK II - TECHNICAL DATA SHEET

LAUNCH:	Fall 1990 from Plesetsk, USSR with Russian launcher
SATELLITE:	"Subtenant" to GEOS, Russian geological research satellite
ORBIT:	circular at 1000km altitude; 83° inclination; orbital period 105 min
PAYLOAD:	linear and regenerative transponder for analoge and digital (AX.25) communications
TRANSPONDER 1:	Uplink: 435.102 - 435.022 MHz (80 kHz bandwidth) Downl.: 145.852 - 145.932 MHz inverted Power: 10 W max. Beacon: CW telemetry (8 parameter) 145.822 MHz ; 200mW digital PSK (30 parameter) 145.952 MHz; 400mW 1100 bit/s scrambled
TRANSPONDER 2:	Uplink: 435.123 - 435.043 MHz (80 kHz bandwidth) Downl.: 145.866 - 145.946 MHz inverted Power: 10 W max. Beacon: CW telemetry (8 parameter) 145.948 MHz ; 200mW digital PSK (30 parameter) 145.838 MHz; 400mW dto. on 145.800 MHz; 2W (1100bit/s scrambled)
RUDAK II:	two on-board computers with IPS operating system for packet radio (AX.25) and digital signal processing (DSP).
HARDWARE:	65C02 CPU with 56 kByte RAM (RUDAK I primary computer) RTX 2000 CPU (RISC 10 .. 15 MIPS) with 128 kByte + 64 k for EDAC at 35ns access time. 1 MByte RAM DISK for Mailbox and data storage Hard command decoder (PN) and baseband switch 4 RX, 1 TX, Modems and 4 DC/DC converter

SOFTWARE: Multiconnect Mailbox (1 MByte RAM), ROBOT mode and Digipeating
IPS Operating System

UPLINK	RX-1	RX-2	RX-3a	RX-3b	RX-4	Unit
Frequency	435.016	435.155	435.193	435.193	435.041	MHz
Baudrate	1200	2400	4800	9600	DSP	bit/s
Modulation	FSK	BPSK	RSM	RSM	any	
Coding	NRZIC Bi-Ø-M	Bi-Ø-S	NRZIC Bi-Ø-M	NRZI NRZ-S + scrambler	I + Q	

DOWNLINK 145.983 MHz with 3W typical (10W optional)

Mode 1:	1200 bit/s; BPSK, NRZI (NRZ-S) (like FO 20)
Mode 2:	400 bit/s; BPSK, Bi-Ø-S (like OSCAR 13 beacon)
Mode 3:	2400 bit/s BPSK, Bi-Ø-S (planned for OSCAR 13)
Mode 4:	4800 bit/s RSM, NRZIC, (Bi-Ø-M) (like 4800 bit/s Uplink)
Mode 5:	9600 bit/s, RSM NRZI (NRZ-S) + scrambler (like 9600bit/s Uplink)
Mode 6:	CW keying (for special events)
Mode 7:	FSK (F1 or F2B) e.g. RTTY, SSTV, FAX etc. (for special events)
Mode 8:	FM by D/A converted signals from the DSP-RISC processor (e.g.speech)

TABLE 1: Main system parameters

ACKNOWLEDGEMENTS

It must really be stressed that the work reported here has been performed by a highly motivated and highly skilled team in less than six months (6.89 - 1.90) starting from paper to flight hardware in "leisure" (hi) time after work. Special thanks to the Marburg based AMSAT group headed by Dr. K. Meinzer, DJ4ZC and W. Haas, DJ5KQ and the RUDAK group DL2MDL, DG2CV, DK8CI, DF8CA, DF2IR and DB2OS.

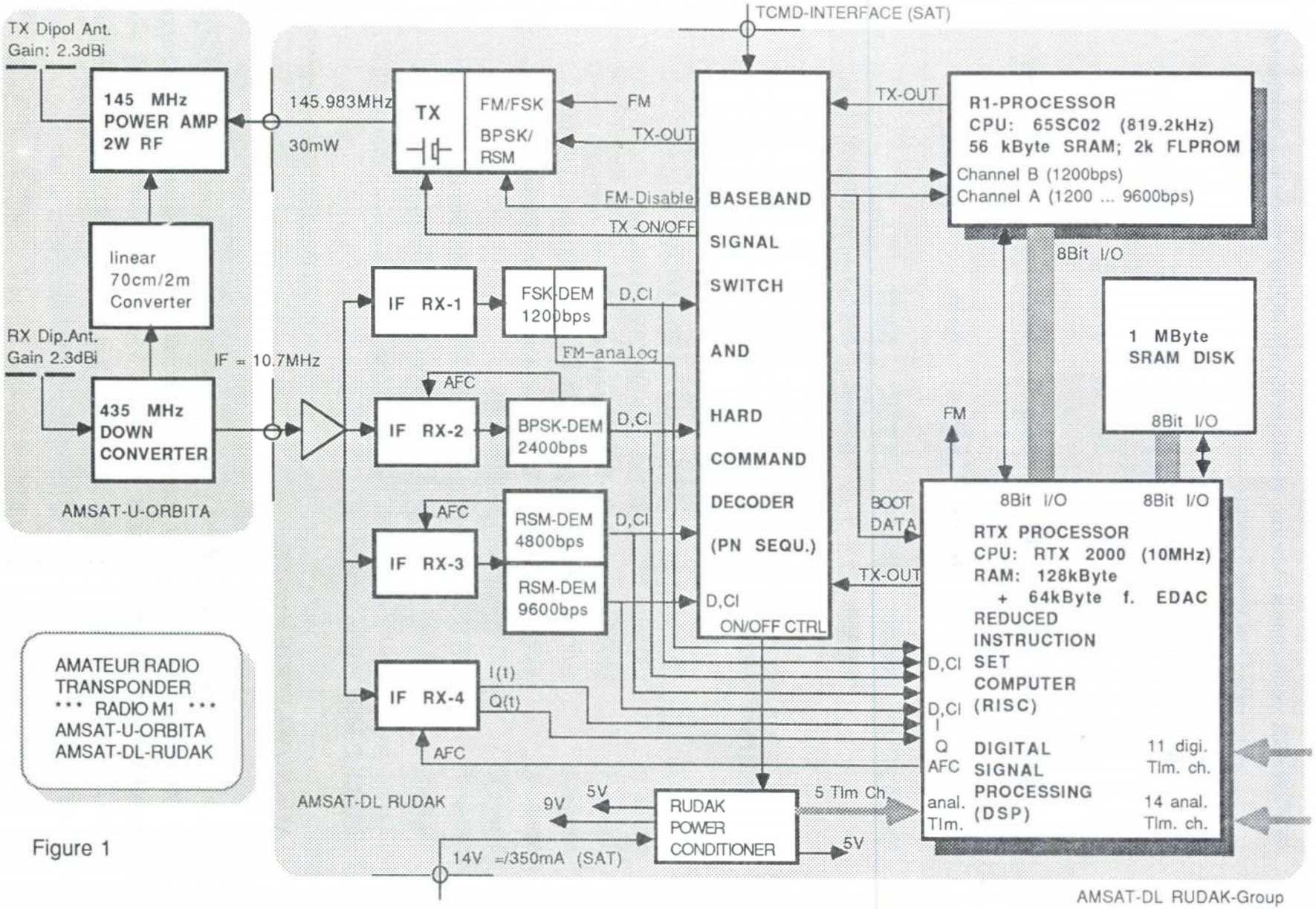
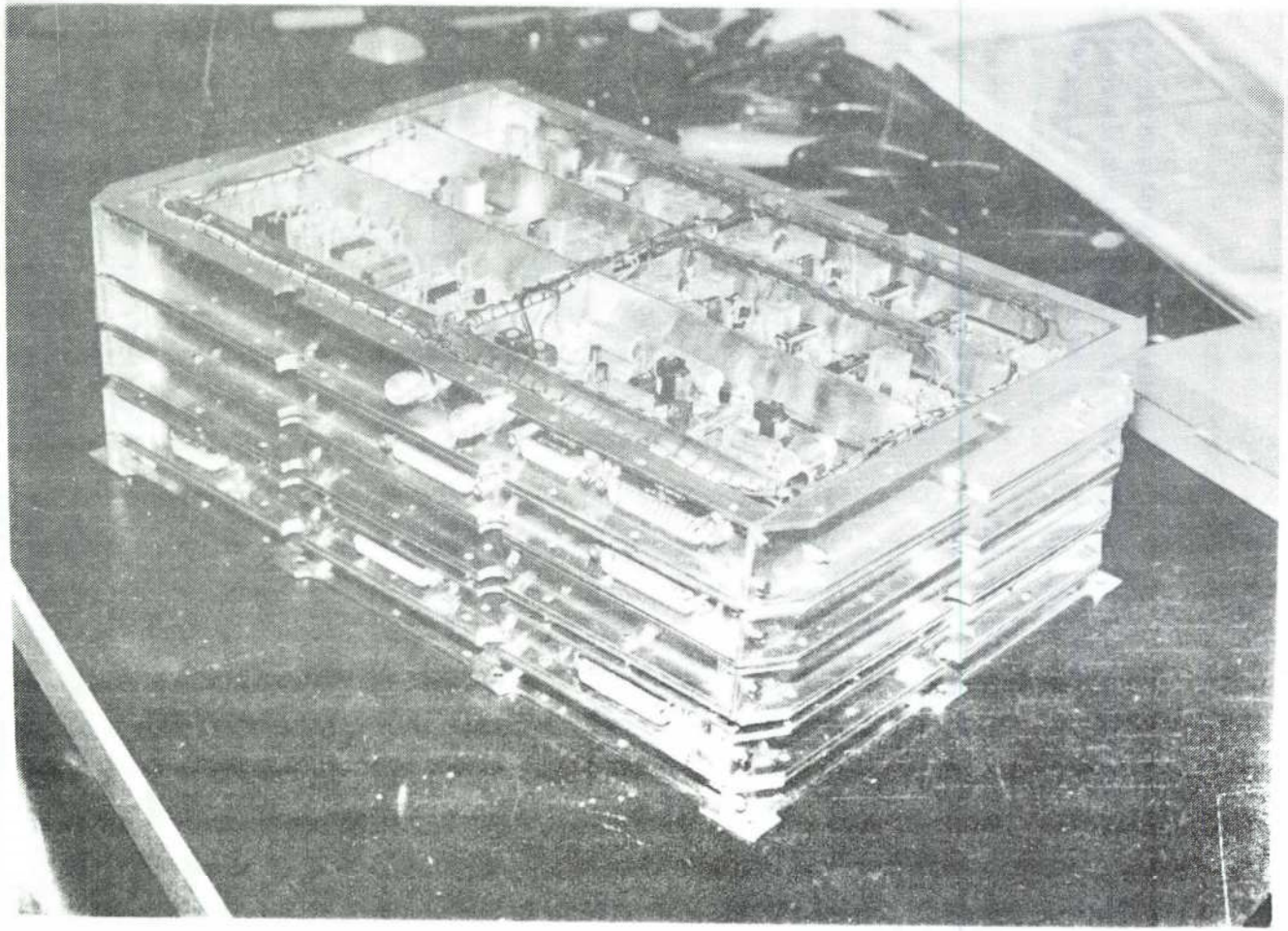

TCMD-INTERFACE (SAT)
TX Dipol Ant.
Gain: 2.3dBi
145 MHz
POWER AMP
2W RF
145.983MHz
30mW
TX
FM/FSK
BPSK/
RSM
FM
TX-OUT
FM-Disable
TX-ON/OFF
BASEBAND
SIGNAL
SWITCH
AND
HARD
COMMAND
DECODER
(PN SEQU.)
ON/OFF CTRL
TX-OUT
R1-PROCESSOR
CPU: 65SC02 (819.2kHz)
56 kByte SRAM; 2k FLPROM
Channel B (1200bps)
Channel A (1200 ... 9600bps)
8Bit I/O
1 MByte
SRAM DISK
8Bit I/O
linear
70cm/2m
Converter
RX Dip.Ant.
Gain 2.3dBi
IF = 10.7MHz
435 MHz
DOWN
CONVERTER
AMSAT-U-ORBITA
IF RX-1
FSK-DEM
1200bps
D,CI
FM-analog
AFC
IF RX-2
BPSK-DEM
2400bps
D,CI
AFC
IF RX-3
RSM-DEM
4800bps
RSM-DEM
9600bps
D,CI
IF RX-4
I(t)
Q(t)
AFC
D,CI
FM
BOOT
DATA
TX-OUT
8Bit I/O
8Bit I/O
RTX PROCESSOR
CPU: RTX 2000 (10MHz)
RAM: 128kByte
+ 64kByte f. EDAC
REDUCED
INSTRUCTION
SET
COMPUTER
(RISC)
D,CI
I
Q
AFC
DIGITAL
SIGNAL
PROCESSING
(DSP)
11 digi.
Tlm. ch.
anal.
Tlm.
14 anal.
Tlm. ch.
AMATEUR RADIO
TRANSPONDER
*** RADIO M1 ***
AMSAT-U-ORBITA
AMSAT-DL-RUDAK
AMSAT-DL RUDAK
9V
5V
RUDAK
POWER
CONDITIONER
5 Tlm Ch.
5V
14V =/350mA (SAT)
AMSAT-DL RUDAK-Group

Figure 1



Rudak II Flight Unit

CELP HIGH-QUALITY SPEECH PROCESSING FOR PACKET RADIO TRANSMISSION AND NETWORKING

A. Langi and W. Kinsner, VE4WK

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada R3T-2N2
E-mail: VE4WK@VE4BBS.MB.CAN.NA

Abstract

This paper presents a design of a signal processing system for telephone-quality speech transmission through a packet-radio network at rates as low as 4800 bit/s in either real-time or off-line mode. A standard 64 kbit/s speech signal is compressed down to 4.8 kbit/s using the Code-Excited Linear Predictive (CELP) coding scheme adapted from the proposed U.S. Federal Standard FS-1016. The CELP algorithm is implemented using a floating-point Digital Signal Processor (DSP) to achieve a real-time, interactive (full- or half-duplex) or fast off-line network-based application. An implementation using a NEC 77230 PC-based DSP Evaluation Board (EB-77230) is under development.

1. INTRODUCTION

High-quality speech transmission over long distances is a problem of considerable importance in voice communication research. In the past, this research and implementations have focused on the transmission of analog signals. However, the quality of analog voice deteriorates rapidly over long-distance VHF/UHF transmission and usually becomes too noisy after only a few repeaters. Consequently, transmission of digital signals is considered superior to its analog counterpart due to the ability to reconstruct the signal at each repeater completely, thus achieving a higher immunity to noise [Kins89]. In addition, the digital transmission can use error protection schemes that increase the robustness of the transmitted signals.

The digitized Pulse Code Modulated (PCM) form of telephone-quality speech requires 64 kbit/s (the bandwidth of 300 to 3300 Hz requires sampling of the signal at 8000 samples per second, and the dynamic range of the signal requires 8 bits or 256 quantization levels). This high bit rate calls for a wider channel bandwidth than is available on most of the HF and VHF amateur bands (6 kHz). Therefore, speech compression techniques are necessary to reduce the bit rate to a level that could be accommodated by the channel bandwidth. It should be noted that if the bit rate is sufficiently low, then more than one voice can be transmitted in a single channel. Speech compression also reduces the storage required in the store-and-forward communication mode. Speech compression may also be used for voice encryption and decryption in secure voice communications.

One of the important modern speech compression techniques is the Code-Excited Linear Predictive (CELP) coding. The CELP method is outstanding in that it produces a high-quality speech that is better than any of the presently used U.S. Government standards at 16 kbit/s, and is comparable to 32 kbit/s CVSD (Continuously-Variable Slope Delta) modulation, at rates as low as 4800 bit/s [CaTW90]. Since it is becoming a new U.S. Federal Standard (FS-1016), it is expected to be used widely.

This paper presents a design of such a speech compression system for packet radio, using a PC-based NEC 77230 Evaluation Board [Soni87]. This work is an extension of our 2.4 kbit/s real-time speech research projects using special-purpose DSP processors, bit-slice microprogrammable processors, and multiprocessors.

2. SPEECH PROCESSING

2.1 Speech Coding for Packet Radio

Methods for digital speech coding can be categorized into two types: waveform coding and source model coding [KlKi87, MiAh87]. The waveform coding techniques imitate the waveform as faithfully as possible, thus producing high-quality speech at the expense of higher bit rates (above 10 kbit/s). Examples of this coding are the PCM, Adaptive Differential Pulse Code Modulation (ADPCM), Delta Modulation (DM), and Adaptive

Predictive Coding (APC) [KlKi87].

On the other hand, the source model coding techniques imitate the speech waveform production system by finding speech production parameters such as: (i) *spectral (formant) predictor* that represents the shape of the vocal tract, (ii) *gain* that represents the loudness of the speech, (iii) *pitch period* that represents the basic frequency (pitch) of the speech, and (iv) *voiced/unvoiced switch* that determines whether the speech is voiced or unvoiced. This speech modelling produces rates below 10 kbit/s. For example, intelligible speech can be obtained from approximately 2 kbit/s [Swan87]. However, the quality of speech using this coding usually is not as good as the waveform coding because the speech production parameter extraction techniques provide approximations of the vocal tract shape and vocal cord excitation signals [AtRe82]. Examples of this coding are the Linear Predictive Coding (LPC), phase vocoder, and channel vocoder [RaSc78].

The CELP technique [KeST89, CaWT89, CaTW90] can be classified as a hybrid model of the two coding schemes. It is another version of the Multi-Pulse Linear Predictive (MPLP) coding that is an evolution of the classical LPC [Atal86, ScAt85], and is also very similar to the APC which is a waveform coding [CoKK89, ScAt85].

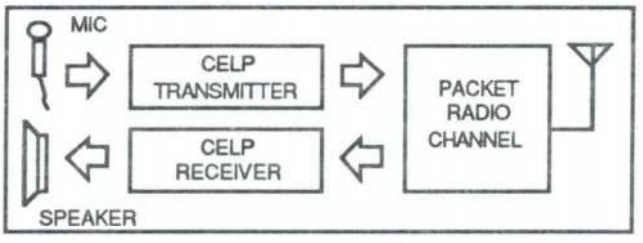


Fig. 1. High-quality speech processing system for packet radio.

A simple CELP speech processing system for transmission using packet radio consists of three parts, as shown in Fig. 1. The CELP transmitter compresses the speech signals into the CELP parameters and protects them using an error correction scheme. The CELP receiver converts protected CELP parameters into audible synthetic speech signals. The packet radio channel transmits or receives the protected CELP parameters to or from the other speech processing system respectively to establish two-way communication.

2.2 The FS-1016 CELP Coder

The FS-1016 standard strictly defines the CELP parameters for transmission, their bit allocations for each frame, their update rates, the bit rate, and frame length, but leaves many options for its implementation.

2.2.1 Speech Transmitter

As shown in Fig. 2, the FS-1016 CELP transmitter consists of the following two main parts: (i) CELP analyzer and (ii) Parameters Encoder, both described below.

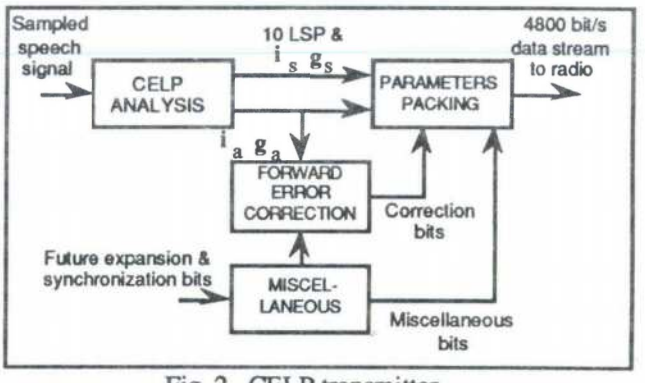


Fig. 2. CELP transmitter.

The CELP analyzer (Fig. 3) maps the speech signals into CELP speech parameters using an analysis-by-synthesis method [ScAt85, KrAt87]. The parameters are: (i) the Linear Predictor (LP) parameters, (ii) the Stochastic Code Book (SCB) parameters, and (iii) the Adaptive Code Book (ACB) parameters. The LP parameters consist of 10 predictors in the Linear Spectrum Pair (LSP) form [SoJu84]. The SCB parameters are SCB entry, i_s, and SCB gain factor, g_s. The ACB parameters are ACB entry, i_a, and ACB gain factor, g_a.

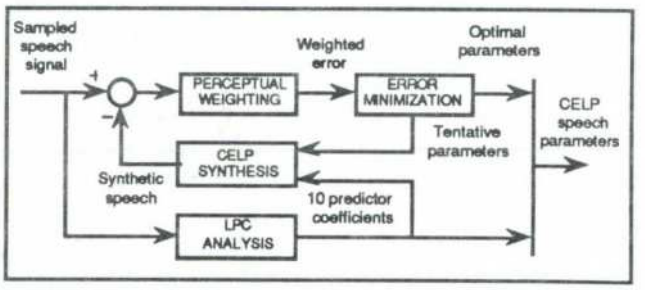


Fig. 3. CELP analysis.

The Linear Predictor (LP) parameters are determined by a standard LPC analysis [Pars86]. First, the digital speech signal is windowed by 30 ms Hamming window without pre-emphasis. The predictors are analyzed using an autocorrelation method. The predictors are scaled to have 15 Hz expansion before being converted into the LSP form.

The other CELP parameters (i_S, g_S, i_a, and g_a) are determined by searching optimal parameters that minimize the perceptual difference between the original and the synthetic signals. The Error Minimization scheme creates tentative parameters to be used by a CELP synthesizer. Then the CELP synthesizer produces the corresponding synthetic speech signals. The synthetic speech signals are compared to the original signals to provide the error signal. The Error Minimization scheme then chooses the optimum parameters that produce the minimum error, and sends them together with the LSP as the CELP speech parameters.

A digital filter called Perceptual Weighting Filter processes the error signal to become a valid measure of the perceptual error [AtRe82]. It de-emphasizes the error that occurs in the formants regions in the error spectrum since larger error in those regions can be tolerated due to the high concentration of energy in those formant regions. On the other hand, it emphazises the error that occurs in the regions between formants because the speech is perceptually sensitive in that region.

As shown in Fig. 4, the CELP synthesizer that is used here has three main parts : (i) SCB, (ii) ACB, and (iii) LP. The SCB, together with the ACB, is responsible for generating the excitation pulses for the LP. The code book entry or index, i_S, is used for addressing 512 code words, and the gain factor, g_S, is used for the gain adjustment of the code words. The parameters are updated four times each frame of 30 ms. The number of bits per frame is 9×4 and 5×4 for i_S and g_S, respectively. Each code word is a sequence of 60 samples. This sequence is created by center-clipping of a Gaussian noise sequence, that causes 77 % of the samples to be 0. Each sample is ternary quantized, so it only has a value of -1, 0, or 1. With those characteristics of the code words, the computation can be much reduced.

On the other hand, the ACB is used to provide the pitch information into the excitation pulses. The code book entry or index, i_a, is used for addressing 256 code words, and the gain factor, g_a, is used for the gain adjustment of the code words. The parameters are also updated four time each frame. The numbers of bits per frame is 8+6+8+6 and 5×4 for i_a and g_a, respectively. The ACB consists of 128 integer delay and 128 non-integer delay values, ranging between 20 to 147, representing pitch frequencies between 54 Hz and 400 Hz.

The LP is a 10-order all-pole filter that represents the vocal tract effects in the speech production. The parameters of the LP are received every 30 ms (1 frame), but they are updated four times every frame by interpolating. The numbers of bits per frame for each coefficient are 3, 4, 4, 4, 4, 3, 3, 3, 3, 3 respectively.

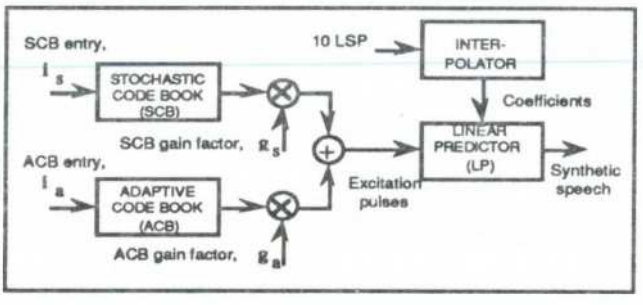


Fig. 4. CELP synthesis.

An interoperable analyzer can be used for reducing the computational complexity. For the SCB, the searching can be done in any subset of the code book interchangeably. For the ACB, a different interoperable way is implemented [CaTW90]. For every first or third (odd) subframe, the normal search is applied into any subset of 128 integer or 128 non-integer delays, and for every second or fourth (even) subframe, delta search is applied and the result is a 6 bit offset relative to the previous subframe delay.

The Miscellaneous block provides a synchronization bit, that is useful to mark the beginning of each frame, and a future expansion bit.

The Forward Error Correction (FEC) scheme protects the most sensitive parameters of the robustness of the speech signals due to the noise in the channels and backgrounds. We only concentrate on the noise in the channel because the CELP is insensitive of the noisy backgrounds due to its waveform character [CoKK89]. Not all the parameters are protected by the FEC because they have different degree of sensitivity and due to the

limitation in the bit allocation and the computation complexity. For a typical CELP, an informal listening test [CoKK89] showed that the 10 LSP parameters were the most sensitive to error, followed by the g_S, i_a, g_a, and the i_S. Only the ACB parameters and the future expansion bit are protected by the the (15,11) Hamming code FEC, while the LSP protection relies on the stability rules in the synthesizer.

The CELP speech parameters are packed with the error-correction bits, as well as synchronization and future expansion bits to create a data stream for transmission through the packet radio channel. Bit allocation for this FS-1016 CELP is summarized in the Table 1.

Table 1. Bit allocation.

TYPE	ALLOCATION		TOTAL
Linear Predictor	3,4,4,4,4,3,3,3,3,3		34
Adaptive CB	Entry : 8+6+8+6	Gain: 5 x 4	48
Stochastic CB	Entry : 9+9+9+9	Gain: 5 x 4	68
Synchronization	1		1
Future Expansion	1		1
Error Correction	4		4

2.2.2 Speech Receiver

As shown in Fig. 5, the FS-1016 CELP receiver consists of three parts: (i) CELP Parameters Decoder, (ii) CELP synthesizer, and (iii) Adaptive Post Filter.

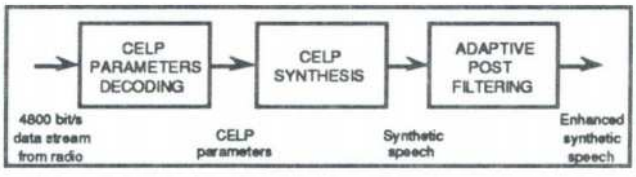


Fig. 5. CELP receiver.

The Parameters Decoder (Fig. 6) extracts the CELP parameters from the data stream. The data stream is unpacked to have: (i) the 10 LSP, (ii) the SCB parameters, (iii) the ACB parameters, (iv) the error-correction bits, (v) the frame synchronization bit, and (vi) the extra future expansion bit. The Stability Rules block is used to ensure that the 10 LSP that have been passed through the noisy channel will construct a stable Linear Predictor. The Adaptive Nonlinear Smoother and the FEC are used to adjust some sensitive CELP parameters due to the error that might happen during the transmission. The Miscellaneous block extracts the future expansion bit and detects the frame synchronization bit to time the decoding and synthesizing process.

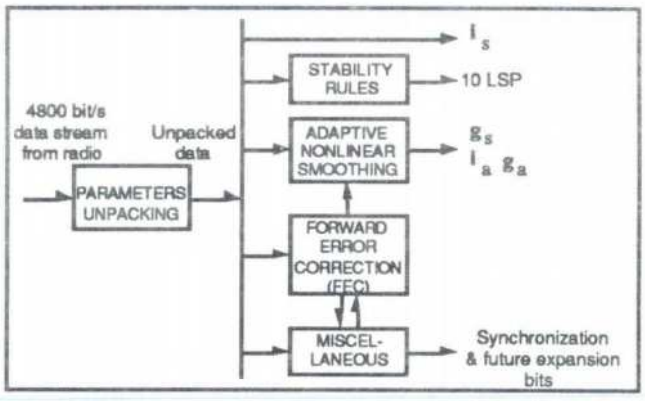


Fig. 6. CELP parameters decoding.

The CELP synthesizer receives the CELP speech parameters and produces a digital speech waveform that can be transformed into audible speech signals using a digital-to-analog converter (DAC), an audio amplifier, and a speaker. The CELP synthesizer used here is a replica of the CELP synthesizer that is used in the transmitter.

The Adaptive Postfilter is useful for signal enhancement. This filter exploits the masking properties of the human ear [ChGe87]. The filter is a short-term pole-zero type with adaptive spectral tilt compensation [CaTW90].

3. A CELP SYSTEM

3.1 Computational Power Requirement

The system shown in Fig. 7 is intended to work in either a real-time or non-real-time mode. In the real-time mode, the system computational speed is very critical. For a CELP system with full-duplex and 256 code words, a DSP processor is required with approximately 17 MIPS (million instructions per second) [CaTW90]. To reduce the overall computational speed, the code books searching computations should be reduced because they dominate the CELP process. This can be done by reducing the size of the code books at the expense of lowering speech quality. Similarly, other functions such as signal enhancing (post-filtering) may have to be eliminated. This reduction in

computation can also be done by using more efficient searching techniques. Other real-time implementations using low-speed DSP chips must be done with multiprocessors.

computational burden. It also includes a development system to make the software development easier.

A microphone is connected to a gain adjuster. The output of the amplifier then is connected to the serial

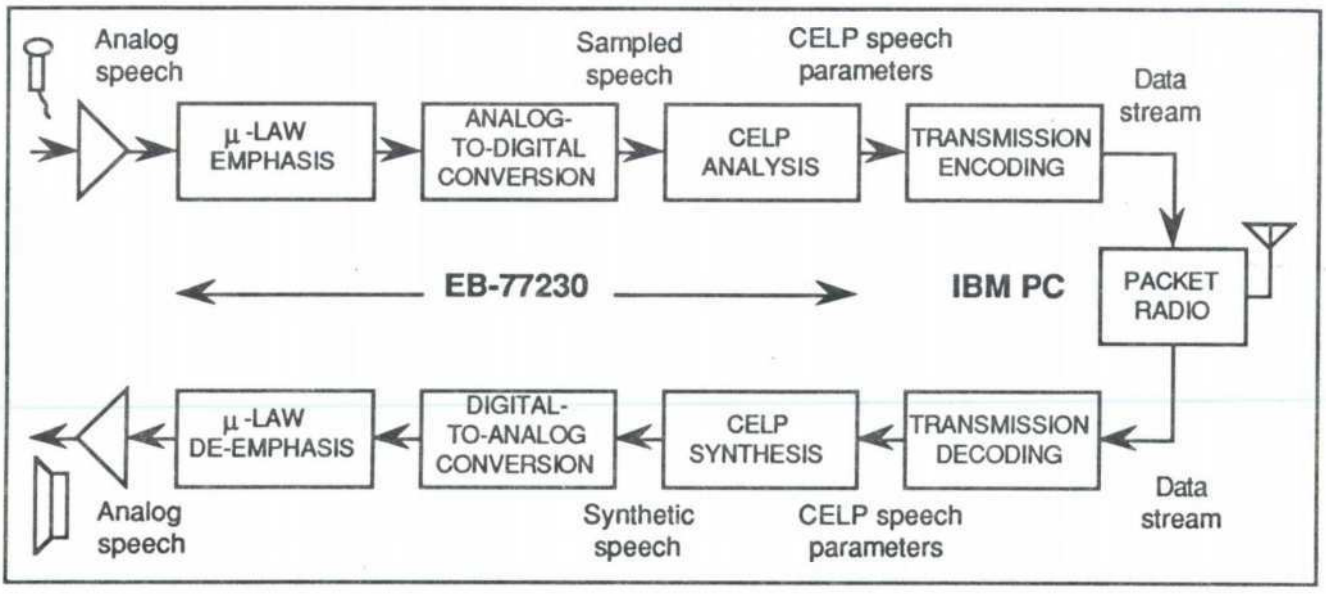


Fig. 7. System configuration.

In the full-duplex mode, the system works at its maximum computational power because the CELP transmitter and receiver must work in parallel. On the other hand, the computational power is reduced in the half-duplex mode because the transmitter and receiver work one at a time only.

The non-real-time mode is common in networks. It reduces the need for high computational power by eliminating some of the functions that deal with the channel noise because many networks provide protocols with error control.

3.2 EB-77230 Implementation

Although this system can be implemented using special-purpose VLSI DSP processors, the EB-77230 add-in board is used to facilitate research and development of a robust speech system for packet radio. It contains a NEC 77230 DSP with 150-ns instruction cycle whose architecture is suitable for voice application. It also has a CODEC which eliminates the development of anti aliasing filter and emphasis circuits. It is an IBM PC-based board that can work in parallel with the PC, thus enabling the PC processor and co-processor to share the input of the EB-77230. The board performs the μ-Law analog compression and decompression, 64-kbit/s analog-to-digital and digital-to-analog conversions, and the most demanding CELP analysis and synthesis. The serial output of the EB-77230 is connected to a loudspeaker through the second channel of the amplifier. The EB-77230 is installed in one of the PC bus slots. The TNC is connected to an IBM PC through an RS-232 cable. The PC performs all the networking tasks, while the TNC in the packet radio is responsible for the modem functions. For real-time transmission, a faster modem (at least 4800 bit/s) must be used [JoFr89].

4. CONCLUSIONS

A design of a high-quality low-bit-rate speech processing system for transmission using packet radio has been described. The design uses an implementation of the powerful CELP speech coding method, based on its new definition contained in the FS-1016 standard. The main problem in implementing a real-time CELP speech system is the very high computational power needed by the CELP algorithm. This is handled in our design by a

DSP subsystem (EB-77230) co-operating with a host computer (IBM PC), using an efficient implementation of the CELP algorithm. Robustness of the system to noise during speech data transmission is achieved through a FEC scheme, as well as the use of stability rules and parameter smoothing. Experimental work is intended to further reduce the computational complexity of the system and to improve error control.

ACKNOWLEDGEMENTS

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, the World University Service of Canada (WUSC), and the Inter University Centre (IUC) on Microelectronics, ITB, Indonesia.

REFERENCES

[Atal86] B. S. Atal, "High-quality speech at low bit rates: Multi-pulse and stochastically excited linear predictive coders," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 3, pp. 1681-1684, 1986.

[AtRe82] B. S. Atal and J. R. Remde, "A new model of LPC excitation for producing natural-sounding speech at low bit rates," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 1, pp. 614-617, 1982.

[CaTW90] J. P. Campbell, Jr., T. E. Tremain, and V. C. Welch, "The proposed Federal Standard 1016 4800 bps voice coder: CELP," *Speech Technology*, pp. 58-64, Apr./May 1990.

[CaWT89] J. P. Campbell, Jr., V. C. Welch, and T. E. Tremain, "An expandable error-protected 4800 bps CELP coder (U.S. Federal Standard 4800 bits/s voice coder)," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 2, pp. 735-738, 1989.

[ChGe87] J. -H. Chen and A. Gersho, "Real-time vector APC speech coding at 4800 bps with adaptive postfiltering," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 4, pp. 2185-2188, 1987.

[CoKK89] R. V. Cox, W. Bastiaan Kleijn, and P. Kroon, "Robust CELP coders for noisy backgrounds and noisy channel," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 2, pp. 739-742, 1989.

[JoFr89] G. Jones and A. Freeborn, "Tuscon amateur packet radio packetRADIO project" *Proc. ARRL 8th Computer Networking Conf..*, pp. 108-113, 1989.

[KeST89] D. P. Kemp, R. A. Sueda, and T. E. Tremain, "The evaluation of 4800 bps voice coders," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 1, pp. 200-203, 1989

[Kins89] W. Kinsner, "Speech recording and synthesis using digital signal processing," *CRRL Convention*, Winnipeg, MB, Canada, 21 pp., August 1988.

[KlKi87] G. Klimenko and W. Kinsner, "A study of CVSD, ADPCM, and PSS speech coding techniques," *Proc. IEEE/Ninth Annual Conference of the Engineering in Medicine and Biology Society*, pp. 1797-1978, 1987.

[KrAt87] P. Kroon and B. S. Atal, "Quantization procedures for the excitation in CELP coders," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 3, pp. 1649-1652, 1987.

[MiAh87] M. J. Miller and S. V. Ahamed, *Digital Transmission Systems and Networks*. Vol 1 & 2, Rockville, MD: Computer Science Press, 1987

[Pars86] T. W. Parsons, *Voice and Speech Processing*. New York: McGraw-Hill , 402 pp., 1986.

[RaSc78] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech*. Englewood Cliffs., NJ: Prentice-Hall, 312 pp., 1978.

[ScAt85] M. R. Schroeder and B. S. Atal, "Code-Excited Linear Prediction (CELP): high quality speech at very low bit rates," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 3, pp. 937-940, 1985.

[SoJu84] F. K. Soong and B.-H. Juang, "Line Spectrum Pair (LSP) and speech data compression," *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 1, pp. 1.10.1-1.10.4, 1984.

[Soni87] Sonitech International, *EDSP-77230 DSP Workstation User Manual version 1.05*. Wellesley: Sonitech International Inc., 276 pp., 1987.

[Swan87] C. Swanson, "A study and implementation of real-time linear predictive coding of speech," M.Sc. thesis, The University of Manitoba, Winnipeg, MB, Canada, 1987.

The PackeTen® System
The Next Generation Packet Switch

Don Lemley, N4PCR
623 Palace Street
Aurora, Illinois 60506
donl@ldhmi.chi.il.us

Milt Heath
75 S. Chestnut Street
Aurora, Illinois 60506

Abstract

This paper examines the current digital bandwidth crisis which is plaguing amateur packet networks, and details a project in the Chicago area which addresses the problem. The PackeTen® project was conceived to provide a "next generation" hardware and software platform capable of satisfying both the current crisis and to allow for many future enhancements. The system described is a working implementation of the long awaited truly "high-speed" packet switch that is available for general use today.

During the last several years, packet radio has become one of the fastest growing segments of amateur radio. In fact it seems that very few of us have not at least given it a try. Since the infancy of this mode, we have seen it grow from simple keyboard to keyboard conversations (similar to early RTTY operations), to store and forward electronic mailing systems, and even simple networking/routing systems.

This means of information transfer under machine control has made electronic mail an accepted part of the radio amateur's day to day operations. As we became more comfortable with the use of digital communications (in applications such as e-mail), we began to see more and more users on our crowded network channels. At the same time, many packet users began to develop an appetite for more sophisticated networking applications. In fact it is now becoming relatively common for individuals to exchange larger amounts of data, in such forms as computer programs, digitized audio and video images, and on-line applications. Applications such as computerized callbook servers, conferencing bridges, and remote file servers, are also being experimented with.

This combination of large numbers of on-line users and sophisticated applications has now produced a sort of crisis within our amateur packet networks. The one thing that all of these network users have in common is that each is a consumer of digital bandwidth within the network. Every day as we see new users on the channels, and new software applications, we also see our networks becoming more overloaded and less enjoyable to use. This problem of inadequate network resources has already begun to reverse the trend of innovation which we experienced in the early 80's. Many of our networks are overloaded to the point

of breaking by the volume of electronic mail alone, and with more traffic being generated every day by a larger contingent of users, there is no end in sight. Instead of growing and improving, our networks are beginning to sag under their own weight.

One reason for the rapid expansion of packet networking via radio is that it represents an area of electronic technology which is open for experimentation. However, these adverse conditions have also begun to stymie the efforts of those experimenters who are trying to provide new applications for the network. For example, it is of little value to have a local conferencing bridge, if the average response time is measured in tens of seconds. The initial excitement over the new application would soon die out because it was too painful a process to be useful or enjoyable. However, if the network was sufficiently responsive, it is probable that this application would find widespread use.

Over the last three or four years, considerable interest has been generated in the idea that it soon might be possible to develop amateur digital networks with the ability to move these large amounts of data in a timely fashion. This idea was based on several factors. The first is that microprocessor technology is becoming increasingly more powerful and less expensive, allowing the development of more advanced networking architectures. The second is that modem technology is becoming available which allows higher data rates while utilizing minimal spectrum. The last major factor in this equation is the development of the increasingly sophisticated software systems needed to support our growing network requirements.

These new technologies promise to provide network system designers with all of the ingredients to build the higher performance networks which are needed today, as well as providing a base for future expansion and improvement.

Surprisingly however, several years after it's introduction in the early 80's, the mainstay of the amateur packet community is the standard 1200 baud TNC-2. Why is it that we haven't advanced to the much superior technologies which are available today? During the above mentioned period, software systems capable of handling new sophisticated applications have indeed become a reality and are in common use today. However, available hardware technology remains a problem.

Modem technology has advanced considerably since the introduction of the TNC-2. We have seen the development of the K9NG 9600 bps modem, and it's variants, as well as the WA4DSY 56k bps radio/modem, and most recently even a system for a full megabit per second data link, as described by Glen Elmore, N6GN, in Ham Radio Magazine, December 1989. These advances have met with limited success for various reasons. However, for systems operating at rates of 56 kb/s, or higher, one of the big problems has been the lack of digital communications hardware capable of handling the higher data rates. In fact, the WA4DSY modem / radio system has been available for over two years, but, with no good solution to the digital problem, it has not achieved the success which one might expect.

Several prototype digital hardware systems have been constructed and tested over the past few years, with varying capabilities, but none have been made available to the general ham community. This situation prompted the authors to design and make available a digital hardware system capable of addressing the packet networking requirements of both current and future systems.

The PackeTen system is designed specifically to address the needs of a high speed data switching node. It was initially conceived to solve the network overload problems in the Chicago area, where the K9NG modem based 9600 bps backbone was, at times, overloaded by the sheer volume of mail traffic alone.

Design Criteria Of The PackeTen® Switch

- Data rates in excess of 1 megabit per second.
- Total aggregate throughput of at least 1 megabit per second.
- Five or more channels.
- Operation from EPROM or RAM.
- A standalone version for "mountain-top" applications.
- Non-volatile memory to support re-boot and auto-configuration.
- An IBM PC XT/AT compatible version with high-speed file access.
- Large code space and data space.
- A port of KA9Q's NOS (latest version of KA9Q's TCP/IP system) in ROM, "NOSINABOX", so the average radio amateur can bring-up the packet switch immediately.

The design goal was to provide a digital hardware and software system capable of handling large amounts of data with a capacity adequate for both now and the forseeable future. Immediate requirements dictated that the system handle at least three 56 kbps links utilizing the WA4DSY radio/modems to implement the "cellnet" concept (described by Don Lemke in the 1987 CNC proceedings) here in the Chicago area. However, since 56 kbps is not sufficient bandwidth for the envisioned networks of the future, we perceived a need to handle several channels running at multi-megabit rates for this system to be the packet switch of choice for the early 90's.

Overall system throughput was of paramount concern. The completed system must be adequate to handle such applications as realtime digitized voice channels, and internetworking of computer resources as well as more typical data communications functions such as electronic mail, file transfers, callsign database servers etc. In order to meet these requirements, the system must be capable of a minimum total aggregate throughput in excess of a megabit per second.

Since it was envisioned that this system would be used as the major internetworking component for our growing needs, it was decided that minimum capabilities would include support for multiple high-speed channels to allow for network growth. Additionally, in order to provide access to the network, the system should provide lower speed user access channels at more conventional rates such as 1200 to 9600 baud. After some debate we settled on a 10 channel system providing 6 high-performance networking channels and 4 local access ports.

Another concern was the necessity to provide adequate system memory resources. In order to develop sophisticated software systems with large program and data memory requirements, the hardware must provide a healthy complement of memory in the minimal configuration with lot's of expansion capability built in for the future. Also since the system was to have a standalone configuration, some form of non-volatile storage was required. (If the standalone system resets (e.g. a power failure, etc.) the non-volatile memory is used to auto-configure the system at startup.)

Another major design consideration was that the system operate from a single 5v supply, and require very small amounts of power to operate so as to be practical for use at remote sites under adverse conditions such as a solar powered site. This implied a CMOS design throughout.

Implementation

During the course of our development, various different hardware prototypes were constructed and tested. Original experimentation focused on an Intel/Zilog architecture, and in fact prototype systems with conventional processors and DMA channels were built and tested for use in this application. Results of this approach were positive, producing systems capable of multiple channels of 56 kbps full duplex serial data, but insufficient for our design goals.

The breakthrough for this project came with the announcement by Motorola of their new MC68302 processor. This processor contains a 16/20 MHz 68000 processing core, and an additional RISC based communications processor (CP). The part contains 3 high speed serial channels and complete full duplex DMA support for those channels, allowing operation at baud rates well in excess of our stated megabit per second goal. In fact, the MC68302 can support bit clocking rates as high as 6 Megabits/sec within certain overall throughput constraints. The CP provides microcode to support five link level protocols including HDLC. Additionally this highly integrated processing platform provides several hardware timers, prioritizing interrupt controllers, dual port memory, etc. It is implemented in CMOS, so that it's power requirements are small (approximately 50 ma during full operation), and in it's lower power/standby modes the power requirements are miniscule. This single chip was designed to fill virtually all of our design goals and provides in a single highly integrated package both a powerful general purpose processing platform and a dedicated high performance communications processor.

The MC68302 processor then became the platform for the development of the PackeTen® switch. To provide local/console access we chose the venerable 85C30

SCC. This brought the total number of channels to five: 3 multi-megabit capable channels and 2 local access channels. To meet our 10 channel criteria required two MC68302's and two 85C30's.

During the initial design phases one of the factors taken into consideration was the need for modularity and expandability. Since a full-blown system would not be necessary in all applications, the system needed to be scalable to its application. This factor prompted us to separate the 10 channel system into a 5 channel IBM PC compatible card and a 5 channel standalone with the unique capability to connect the standalone card to the PC card and thus meet our goal of a 10 channel packet switch. This capability would provide the network builder with the system he needs now as well as expandability for future network requirements. The result is a 5 channel PC system (main card), a 5 channel standalone (daughter card), and by combining the two systems a 10 channel PC system.

Each system can support from 128k to 1 Megabyte of EPROM. Our current version of KA9Q's NOS requires approximately 300k of code space, allowing ample room for expansion. Each system provides for up to 2 Megabytes of 0 wait state RAM allowing large systems sufficient buffer space to avoid data loss during network congestion.

Each processor has 2k of EEPROM for initial configuration storage. Typical information stored in EEPROM is as follows

1. IP Address.
2. Call sign.
3. Host name of node.
4. Default route.
5. Default attach command.
6. Site alias name.
7. System operator password.
8. Optional NOS command lines which can be saved in EEPROM. These command lines are used to store additional, site specific configuration parameters.

Another major part of the PackeTen® system is the 16-64k of tri-ported memory. With a 10 channel system all three processors can communicate with each other via inter-processor communications channels (IPC channels), which are attached just like any other type of communication channel in NOS. Utilizing this approach the PackeTen processors communicate with the host processor (PC) and with each other through virtual network connections or "IPC" shared memory "links". When configured in this manner the PackeTen system becomes a networking co-processor offloading network traffic handling from the PC. This

allows the PC to be used for other applications. while the PackeTen remains a fully functional IP switch node in the network. For example, traffic arriving on one of the daughter card's channels can be routed through an IPC "link" to the main card for transmission through one of the main card's ports. Also note that while the example given here is for a "PC" version of the system, it also applies directly to the "standalone" PackeTen system when configured as a 10 port remote packet switch.

Since the PackeTen system was intended to provide a reasonably long term solution to the digital networking problem, it was decided that the physical network interfaces should be configurable in order to accommodate possible future modem/radio interface standards. The chosen solution was to separate the physical link interfaces from the main processing platform. Immediate needs dictated that an RS-232 interface be provided to support existing TNC's and modems for the local access channels, and a TTL level interface for such modems as the WA4DSY radio/modem. Therefore a combination RS-232/TTL interface card was designed. In addition to simple level conversion functions, this card provides commercial grade features such as digital loopbacks on the links to provide for extensive diagnostics, and individual programmable control lines for radio /modem control functions. Additional interface cards are currently under consideration for such standards as RS-422 and ECL interfaces.

Software

No discussion of the PackeTen® system would be complete without at least some mention of the software system which supports it. As mentioned previously, one of the problems with building good networks is the requirement for more sophisticated software systems. From the beginning, we intended to produce a complete digital networking solution, and we believed that the place to start was with the KA9Q TCP/IP NOS networking package. However, in order to take advantage of this system, much work had to be done to: 1) Port the software from an INTEL based platform to a Motorola 68k system, and 2) Develop the software necessary, to allow NOS to work in a standalone/diskless configuration (i.e. NOSINABOX). The culmination of this effort is what we affectionately refer to as NOS302. This package provides the user with a complete implementation of NOS in EPROM, which will allow the user to take advantage of the PackeTen hardware, and put that system into service today. The list that follows provides insight into some of the changes neccessary to support a standalone NOS system.

- Support to provide traditional AX25 users with PAD functionality, and access to the IP network. This feature is provided in standard NOS through the "mailbox" feature, but required much rework in order to provide the PAD functions in a standalone/diskless environment. The PAD function was deemed extremely important in order to allow non-IP users access to the network facilities without the need for a PC running an IP implementation. By providing this function, the packet user with an off the shelf TNC can also use the network effectively.

- Comprehensive on board diagnostics.

- Non-volatile configuration memory support.
- IPC shared memory virtual "links" between processors.
- Improved data buffering/queueing mechanisms for high speed operation. "Scatter gather" transmit and receive buffer chaining.
- Interrupt driven software timers added to the kernel allowing precise timing for such driver functions as keyup/keydown of transmitters without the use of polling loops which waste processing power.
- Synchronous drivers providing full/half-duplex, DMA support for external high speed HDLC modems. These drivers support full clocking options both internal and external for receive and transmit clocking.
- Asynchronous drivers for local terminal/TNC communications, SLIP links, etc. DMA and interrupt driven configurations.
- Full support for all currently supported NOS protocols, including AX25, SLIP, SLFP, NETROM, NRS, etc.

Status

The results of the PackeTen® project have been quite good. Prototype system tests were completed early this year, with production grade systems consisting of 6 layer PC boards available now. These systems are supplied complete with software and are available through Grace Communications, Inc.

Current installations are in service in Youngstown, Ohio, as well as in and around Chicago, with others soon to come on-line in other states. The Chicago based systems are being used to implement full-duplex 56 kb/s "cell" sites in the construction of a replacement network for our old 9600 baud backbones with excellent results.

Performance indications are also very positive, with systems demonstrating the ability to handle 6 links at 56 kbps, 2 links at 9600 and 2 links at 1200 with no degradation in throughput. Additionally, systems have been successfully tested with a 2 megabit/s links running full duplex.

Conclusions

The PackeTen project has provided a next generation platform for amateur networking in the Chicago area. While this platform undoubtably has provided a major improvement in the state of our networking capabilities, it's main contribution is in it's ability to allow further development of network application systems. Specifically, by allowing developers to work with a more powerful hardware and software platform, they can begin to build the kinds of applications mentioned earlier in this paper, (including digitized voice, remote file access systems, conference bridges, etc.) with the knowledge that sufficient bandwidth should be available to make those systems practical.

The BPQ Node in an Expanding Network

by Karl Medcalf WK5M, and Phil Anderson WØXI,
in cooperation with John Wiseman G8BPQ

July 20, 1990, Lawrence, Kansas

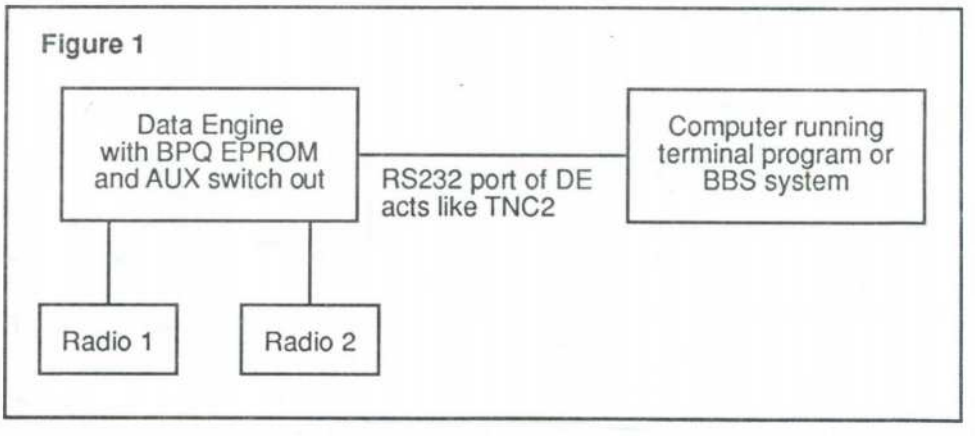
The release of the Kantronics Data Engine, along with the development of their 9600 baud (G3RUH compatible) modem has now added a new dimension to the meaning of a Network Node. With the open architecture of this packet controller, and the flexibility now possible, network expansion and high speed operation is now possible with future expansion in mind.

John Wiseman (G8BPQ) has completed development of his packet switch software in an EPROM version, suitable for installation in the Data Engine. The flexibility written into this EPROM allows you to configure your node to meet your specific requirements. First, let's look at some of the specific features of the BPQ node. This is not a new node, as the G8BPQ packet switch software has been running on numerous systems in its PC based version. Using it in this fashion allows any KISS mode TNC to be a Network Node, providing all the features of Net/Rom nodes, and a few new features.

One of the major advantages of the BPQ node over the other nodes is the "stay" option when connecting to another node or an end user. When this option is specified in the Connect command, the node will not cause a complete disassembly of the network link when the far end of the connection initiates a disconnect. As an example, if you connect to a node named "KSLAW", and then issue a connect of the form "C KSWCH S", this will activate the stay function and connect you to the KSWCH node. From this point you could connect to an end user, or a PBBS near the KSWCH node. Since the stay option is in effect, when you are finished with the PBBS you simply use its BYE command, causing the PBBS to disconnect. When that disconnect is received by the KSLAW node, you will not be disconnected, but you will be sent a message "Returned to node KSLAW". You may now issue further commands to the node.

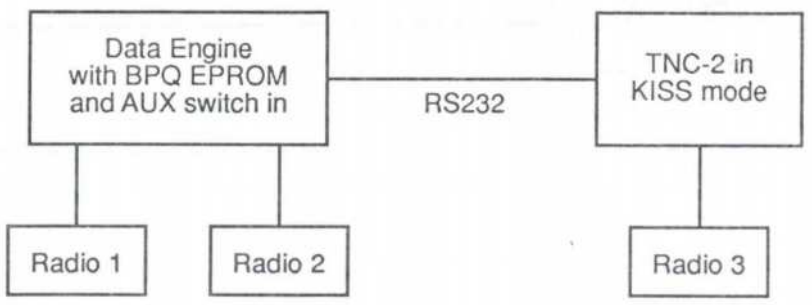
Since the BPQ node software has previously run only in a PC computer (normally co-located with a BBS system) this meant that the node required a computer to be connected and running at all times. Now that the code has been made available in an EPROM version for the Data Engine, the computer is no longer required. Another advantage of this implementation is that since the Data Engine is a dual-port unit, you can now operate a simple two-port Network Node using a single Data Engine with the BPQ EPROM code installed. The serial port of the Data Engine can be configured to appear like a TNC-2, allowing regular use with a simple terminal program, or access by a BBS system, just as though the BBS were connected to a TNC-2. (See figure 1)

Figure 1



Frequently, we have the need for more than two radios in a single node, and this has been cumbersome in past implementations of node firmware. With the BPQ code running in the Data Engine, a simple press of the front panel AUX switch changes the serial port from a TNC-2 port to a KISS link, which may be connected to any KISS mode TNC. In this configuration (see figure 2), you can have a three-port Network node with a single Data Engine using the BPQ code and a TNC-2 in KISS mode. No specific requirements exist for the TNC-2, except that it runs the KISS firmware. Since the Data Engine supports both 1200 and 9600 baud modems, as well as virtually any other known modem, you could configure this three-port node as a gateway for 1200 baud users onto a 9600 baud backbone, or even to a PSK modem for a satellite gateway.

Figure 2

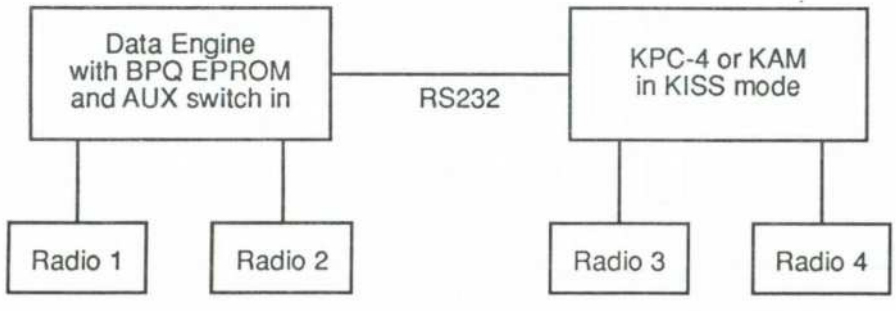


Need more than three ports? Not a big problem with the BPQ/Data Engine combination. A simple 4-port node can be configured using the Data Engine/BPQ combination with a KPC-4 or KAM (KISS mode) connected to the Data Engine serial port. Since the BPQ code allows for dual-port KISS, this provides a low-cost, no-diode implementation of a 4 port configuration. The four ports could all be different radios, bands, speeds and so forth. (See figure 3).

There is another implementation possible due to the incorporation of a new method called "multi-drop" kiss. John Wiseman implemented dual port support using the Kantronics scheme. Since the KISS implementation as originally specified only uses the low nibble (4 bits) of the control byte to specify the function, the high nibble was used by Kantronics in version 2.84 and later of their firmware to address the two ports of the KAM and the KPC-4. If the high nibble was zero, then the unit would address one port, and if the high nibble of this byte was non-zero, it would address the other port.

John has extended this implementation to allow all four bits of the upper nibble to be significant, thus permitting up to 16 different KISS addresses to be defined. He has supplied in his distribution, versions of the KISS code for the various TAPR type TNCs, and documents which byte of this code to change in order to tell each TNC which address it is assigned. Kantronics is also making an EPROM image available for free, non-commercial use, which contains this capability, along with instructions for changing the correct address information. This is available for all Kantronics TNCs direct from the manufacturer's phone-line BBS (913-842-4678).

Figure 3



When using this method, all of the TNCs can be attached to a single serial port, such as the serial port of the Data Engine. A single diode is connected in the RXD line of each TNC, with the anode connected to the TNC and the cathode connected to the serial port of the com-

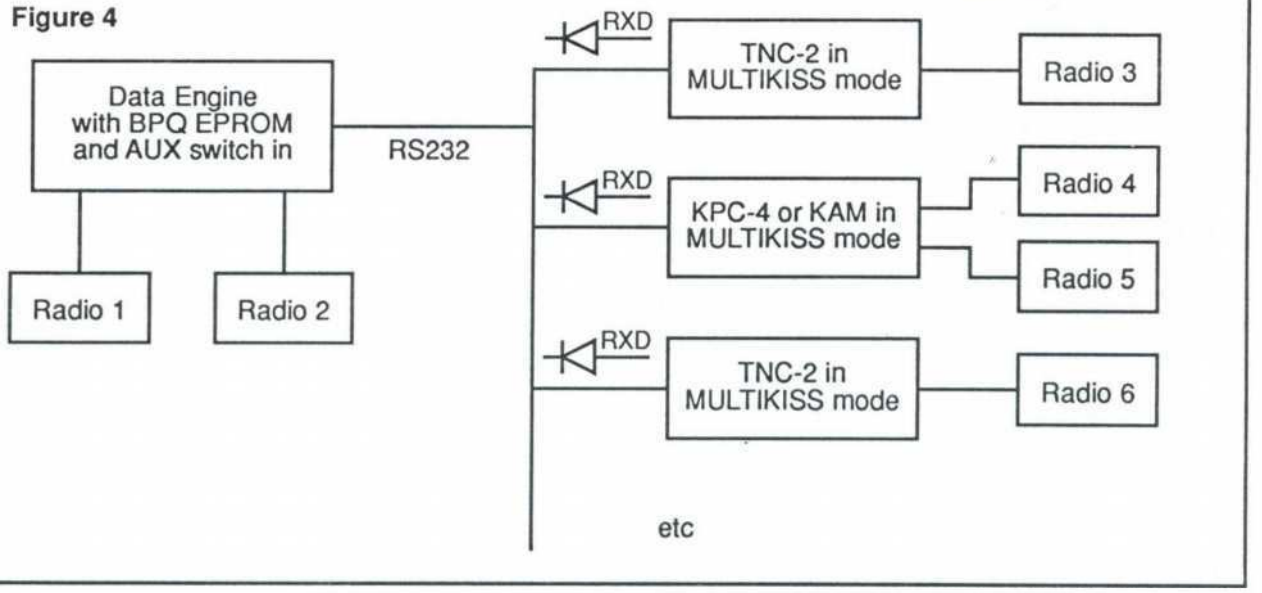
puter or Data Engine. The BPQ code in the computer or Data Engine then polls each TNC in turn (by address) for information, and the TNC then responds with any data that it has received from its associated radio. No data is sent by the TNCs to the host computer or Data Engine without being polled, so there is no conflict in data on the serial line.

Using this multi-drop kiss method, a possible configuration for a multi-port Network Node could look similar to the one shown in figure 4. Using this method, it is theoretically possible to have one Data Engine with BPQ code (2 radio ports) connected to as many as 16 other TNCs using the multi-drop KISS code, each connected to its own radio. You could conceivable mix and match all makes of TNCs, including the KAM and KPC-4 (which each would use 2 addresses and connect to 2 radios).

In this type implementation however, I suggest connecting the busiest radio channels directly to the Data Engine radio ports, since all other TNCs are polled for data, thus possibly slowing them down slightly.

This seems like a possible way to finally begin to improve our Network efficiency at a reasonable cost, and maintain the flexibility to expand to even higher speeds or new modems as they become available. The open architecture of the Data Engine will certainly lead to the development of other firmware/hardware, and perhaps we can again make packet a viable means for communication in real-time as well as a messaging system.

Figure 4



NODE NETWORKING

Donald R. Nelsch, K8EIW
2545 N. Haven Blvd.
Cuyahoga Falls, OH 44223

ABSTRACT

With the advent of NET/ROM (tm) Software 2000 Inc. and NORD<>LINK TheNet software, amateur packet radio networking became possible. However, some of the "factory standard" options in the software tend to limit performance of the system. This paper deals with some of the networking problems the author has resolved, plus some of the hardware considerations that were made in establishing and maintaining the authors' 21 transmitter interconnected node network.

INTRODUCTION

There were (and are) several "players" and prime movers in the N.E. Ohio Packet network, and certainly several interconnecting node operators who have joined us in what I believe to be the nations longest - "high-speed" (if you can consider 4800 baud as "high-speed"!), complex amateur radio packet network. The 4800 baud UHF backbone network as it is now configured, stretches from Cleveland on the north to Lexington KY (and perhaps beyond as you read this) on the south with branches extending into Fremont in N.W. Ohio and to Cambridge in S.E. Ohio. In addition, there are many collocated tributaries on other bands which extend the range and function of the network to many other locations.

HISTORY

I first set up a single NET/ROM (tm) node in Cuyahoga Falls, OH in 1987 and with some grief, was able to "network" to a few distant nodes, but the results were not always the best. As my enthusiasm for packet radio increased, so did my investment in packet radio nodes in several locations in N.E. Ohio. Before long, I had in addition to the node in Cuyahoga Falls (Akron ID), node sites in Mt. Gilead, Cleveland, Canton, Wayne County (Wooster) and most recently, Mansfield. It became apparent early on that we needed a better way to move traffic from city to city than on a heavily used 145.01 MHz. "keyboard" channel, so the idea of "back-boning" the data on a UHF "trunk" frequency was born, and implemented. Initially, our efforts on UHF were 1200 baud, but it soon became obvious

that we needed something faster. The 4800 baud H.A.P.N. modem boards were available and seemed to work with the radios we had and so, an upgrade was planned. We determined very quickly that mixing speeds on one RF channel was an invitation to disaster in terms of interference primarily due to lack of proper data carrier detection between modulation types, so the conversion to 4800 baud on the UHF backbone was accelerated.

Figure 1 shows the UHF portion of the Ohio Network as of July 1990.

HARDWARE SELECTION

One of the early objectives in establishing the network was that any hardware needed to be uncomplicated, easy to replace, connectorized, and interchangeable! To accomplish this objective, I chose the Kenwood TM-221/321/421 series of radios as they were small, reasonably reliable, frequency agile (synthesized), and had the advantage of having fixed level audio available at the mike jack. This meant that we needed only one cable from the TNC to the radio, no pig-tails and no audio pots to tweak (and leave wrong!) The TNC chosen was the MFJ-1270b as it was the least cost TAPR-2 compatible TNC on the market. The TNC is very easy to convert for node operation.

At this point, with very few exceptions, the K8EIW/WB8BII network is modularized to the point that one spare radio per band and one TNC and one power supply is all that is needed to effectively replace any of the 21 nodes should there "minor" flame-out at any of the sites. As one who has been involved with repeaters from the very early days, it is certainly a relief to not have to disconnect and reconnect 20 or more individual wires to replace a piece of equipment!

NETWORKING PHILOSOPHY

The network as I have configured it consists of three basic elements.

1. User Ports.

First, users need access to the network. By tradition, most amateurs have started out on 2 meters with their TNCs, so the logical choice for a "user" port is on 2 meters.

2. High Usage Ports.

Second, there are some high-volume users, such as Bulletin Board/message Systems (BBSs), and PacketCluster DX message systems. As their usage is quite high, they would be an unfair user of the available channel time compared with the individual users. We encouraged the BBSs to utilize frequencies in the 220 MHz band to enter the network for forwarding purposes. I am aware of 9 "Major" BBSs and DX PacketClusters in N.E. Ohio that are making good use of 220 MHz. Are you listening, F.C.C.???

3. Inter-Node Trunks.

The third element, is the inter-node UHF backbone channel which is the main workhorse of the network.

Into this mix, we need to stir frequency selection of both the user ports and the backbone ports. This also affects one other "little" concept of reliability/redundancy vs. the problem of frequency congestion.

In N.E. Ohio, propagation and terrain is such that with a 6 element beam antenna on a 25 watt radio on UHF, we can expect a solid path between the node sites which are between 20 and 45 miles apart, and a useable path most of the time over a 45 to 70 mile path, which is about the distance between two non-adjacent nodes. For purposes of redundancy, I have chosen to put all UHF nodes on the same frequency, and by proper selection of the p-persistance parameter, have managed to get the nodes to "live" together very well. I have found that data transfer is greatly enhanced if a backbone node "talks" only to the immediate adjacent node in serial fashion as opposed to hopping over adjacent nodes. To add to the complexity of this situation, at most sites, I have a 220 MHz node and a 2 meter node that can also "talk" to the adjacent or perhaps the second adjacent node site. It is here that proper selection of node parameters becomes very important to the smooth operation of the network.

ROUTE QUALITY VS NODE QUALITY

Distant nodes often tend to come and go with propagation (and other reasons!), but an adjacent node, hence the ROUTE to that adjacent node tends to remain reasonably stable. If NODES are locked in the data base, it is not uncommon for routing loops to be established when propagation changes, and for whatever reason to not be unlooped when things get back to "normal". It is for this reason that I have chosen to lock

the ROUTE values to known good nodes to a predetermined high value and to default itinerant nodes to some low value such that they will show up in a node and/or route list but not propagate through the network as a first choice route. This saves routing loops and at the same time, also gives the users some idea of what the band conditions may be (is it "up" or "down"?). Also, by proper selection of ROUTE values, the desired path value from any node in the network to any other node (user, high usage or backbone) may be preselected numerically, and will change dynamically as nodes or propagation changes without any human intervention. This means that if an intermediate node is "lost" due to a failure, the routing algorithm in the nodes will cause the routing to try the second choice route, which may be either to "hop" the silent node or possibly to use an alternate frequency to get to the adjacent site.

The point of this exercise is that the node operator's first objective should be to get the data through on the primary route. Should the primary route fail, the second choice route should be a viable path, and if that has also failed, the third choice route is the last hope! This all needs to be done dynamically by the TNC processor and WITHOUT operator intervention. Proper ROUTE quality selection achieves this objective.

ROUTE QUALITY NUMBERS

One of the concepts that needs to be understood by node sysops is the concept of route quality. Basically put, there is an arbitrary "quality" value put on the path between two nodes by the node operator. The quality of a destination node in the originating node list is determined by an algorithm in the firmware based on the route quality. This number is calculated on receipt of each nodes broadcast. The networking routines will select the three best paths to be used in an attempt to establish an end-to-end connection. While the node operators can directly change the path quality to a specific node, it is usually best left to the algorithm to avoid path looping situations.

One of the objectives is to keep all inter-node traffic off the "user" or keyboard channels. By proper assignment of route qualities for adjacent and non-adjacent nodes, we can then determine what the path quality to the node will be, hence, its position on the path list. We can literally force the system to chose an indirect route from one node to another as opposed to a direct route by assigning higher values for the indirect "backbone" routes. In fact, this is what has been done in our network. A 2 meter node will actually chose to connect to an adjacent co-channel 2 meter node via the

backbone route, thereby relieving inter-node traffic congestion on the primary "keyboard" channel. Should there be a failure of the UHF links, the path quality algorithm can then permit a direct 2 meter connection, thereby providing redundancy of path. Likewise, if an adjacent backbone node site is completely disabled, the long-haul logical path can be maintained on the UHF backbone frequency from the first node to the non-adjacent node prior to the two meter path being utilized. To add complication to the maze, in at least three locations, we have a 220 MHz frequency in use as well. Once again, by proper selection of route qualities, we can force inter-node traffic to use first the UHF channel, then the 220 MHz channel, then as third and final choice, the 2 meter path, or perhaps a slightly more round-about path on UHF.

Figure 2 displays the model that I have developed for use in establishing Route Quality Numbers.

One evident fact is that all node operators need to have similar philosophies about "DX" nodes, otherwise convoluted unworkable paths WILL be set up during band openings.

OTHER CONSIDERATIONS

When establishing values for the parameter list, several things need to be considered. First, end to end connections over 3 hops seem to fail when there are more than about 70 nodes in the list. I am sure that the mathematicians in the group can define what that break-point is, but my experience says that anything over about 70 nodes in the list invites disaster. This is especially true if the user attempts the connect using the alias as opposed to the call-sign. If the node list is hard-limited by the parameter list to limit the number to 70, a desired, working, in-use node path could "fall out" of the list when another node broadcast was heard and the list overflowed the in-use node out! Therefore, other means of limiting the number of nodes in the list needs to be made. I have chosen the minimum node quality parameter such that a full-time desired node will have a node propagation distance of approximately six hops. This is usually sufficient for most BBS and TCP/IP purposes. Intermittent nodes, ones that are present only during band openings, will NOT propagate past the initial node stack. This is done by selection of the default node quality (Parameter 3) to be 1 more than the minimum acceptable quality (Parameter 2). Parameter 2 is set sufficiently high so that desired nodes will appear but the number of "normal" nodes will not exceed the approximately 70 number.

There might be a case where a node is a "desired" destination due to its network function, such as a TCP/IP mail server for the region, and is more than six node hops from the most distant entry point to the network. In those cases, the node operator certainly has the option of locking that specific node path value to an artificially high value. In N.E. Ohio, I generally choose the adjacent node to "boost" the path quality of the "desired" node to a very high value. However, should the "desired" node be down for ANY reason, network routing loops will be set up and the "desired" node can expect several hours of no traffic upon its return until the entire network learns of its recovery and/or a sysop intervenes.

As can be plainly seen, path quality determination can be an extremely complex issue and becomes even more so as more nodes are established by more groups. It is certainly not an easy issue when all of the nodes are under one-person control, and is even more complex when there are as many node operators as there are nodes! What is needed is a generalized "standardized" list of parameters for use in a populated network. Perhaps the original "factory standard defaults" were a place to start, but they are absolutely unsatisfactory in todays' environment. Other factors which play an important part in a well-functioning network include p-persistence, time to live, maximum node list, number and frequency of node broadcasts, maximum congestion thresholds, and the list goes on.

Suffice to say, the main objective is to pass the traffic presented in a timely manner with the minimum of congestion created internal to the network. If the nodes are able to communicate with each other on a non-interfering basis during any type of propagation conditions, then we have basically achieved the objective.

It is therefore with a mixture of pleasure and hesitancy that I am suggesting the following list of parameters in Figure 3 as a better place to start than the "factory standard". The hesitancy is knowing that there will be places and conditions that will need different "numbers". I present them as what works in N.E. Ohio with our terrain and propagation conditions, and your conditions may vary "from state to state, local laws prevailing".

ACKNOWLEDGEMENTS

The author gratefully acknowledges the contributions of many people in the development of this system and paper. First, certainly the contribution of my wife, Karen, WB8BII has been above and beyond the call of duty. Her patience with my long hours at the keyboard and on the highway, travelling to node sites, plus the contribution of her call-sign on many of the nodes is greatly appreciated. Second, the site support provided by Marv, W8AZO; Bill, K8SGX; Denny, KN8COQ; Dave, W8GSK; and Bruce, N8ERI has helped immensely. Third, the counsel of Al, K8AL, Vic, K1LT, and especially of Phil, KA8TEF has made this "hobby" project an extremely rewarding personal experience.

Thank you one and all.

Ohio Packet Radio Network

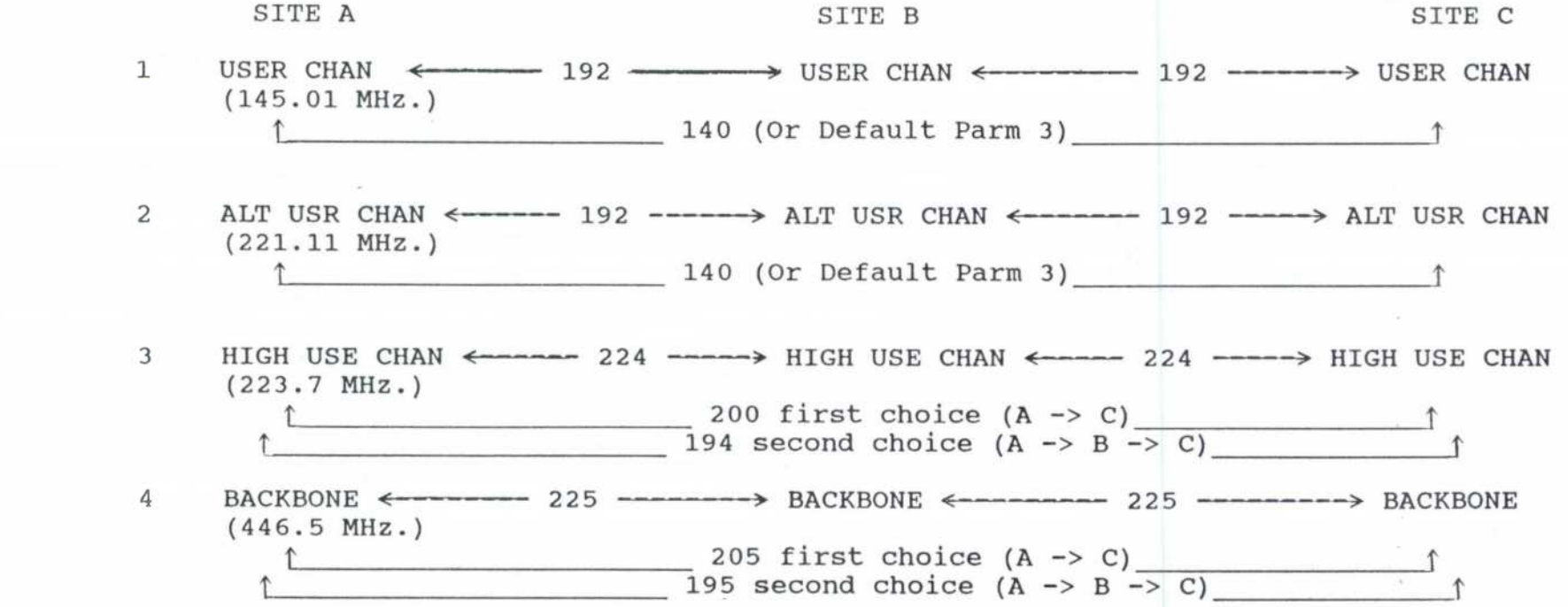
Revised: July 9, 1990

Drawn by K1LT

FIGURE 1


Ohio Packet Radio Backbone
Revised: July 9, 1990
#ABGU:VE3GKM-1
#FREU:KA8TEF-2
#CLEU:K8EIW-8
#AKRU:K8EIW-2
#WAYU:K8EIW-14
#MANU:WB8BII-11
#CANU:K8EIW-6
#MTGU:K8EIW-12
#OLVGN:W8FNW-4
#CMH:K1LT-1
#CMHU1:K1LT-4
#CMHU2:K1LT-7
#HOPWL:NG8P-3
#GNSYU:K8AL-3
#SPGU:WB4LAI-5
#COLU:W8FEH-4
AFIT2:N6KEF-6
#DAYU:WB4LAI-4
YSO:WB8ICL-2
#PERRY:WA8WNI-4*
CYO:AD8I-1
N8XX-3
BKV:N8GTC-2
BKV:N8GTC-4
N8XX-1
#BAIN:NK8T-7
ALMA:K1LT-12*
#CYGU:K8SCH-4
#JAKU:KA8DRR-4
#NKYU:KA4BCD-4
#WPOU:N8BLD-1
#FALU:NK8T-4
446.50 MHz 4800 bps
445.10 MHz 4800 bps
221.11 MHz 1200 bps
*Under Construction
Drawn by K1LT

FIGURE 1A



PATH QUALITIES
(Within Node Stack)

(1 <- 2) = 246	(3 <- 1) = 248
(1 <- 3) = 247	(3 <- 2) = 248
(1 <- 4) = 248	(3 <- 4) = 248
(2 <- 1) = 246	(4 <- 1) = 248
(2 <- 3) = 247	(4 <- 2) = 248
(2 <- 4) = 248	(4 <- 3) = 248

e.g. at [1],	R 1 [2] +246	at [4], R 1 [1] +248
	R 1 [3] +247	R 1 [2] +248
	R 1 [4] +248	R 1 [3] +248

FIGURE 2.

No.	Description	Value	Comments
1	Max. Dest. List	100	See Text.
2	Worst quality for updates	129	Set so that NodeList < 70
3.	Channel 0 default	130	See Text.
4.	Channel 1 (RS-232) Quality	248	See Text.
5.	Obsolescence Counter	6	
6.	Obs. Count. Min.	4	Insures Bdcast is not "trashed" 1st Time
7.	Auto Update Interval	1800	1/2 hr Helps Keep List "Fresh"
8.	Time to live	15	More reasonable!
9.	Transport Time Out (sec.)	120	Give it a chance!
10.	Transport Max. Tries	5	Give it a chance during busy periods!
11.	Transport Ack Delay (sec)	10	
12.	Transport Busy Delay (sec.)	180	
13.	Transport Window Size (frames)	4	
14.	Congestion Control (frames)	4	Don't fill the TNC buffers!
15.	No Activity Time Out	900	Set MUCH Higher for DX PktClstr Access Chan.
16.	P-persistance	50	Max for Backbone - 64 OK for User Channel
17.	Keyup Slot time	10	
18.	Link T1 "Frack" (sec)	4	
19.	"MAXFRAME"	7	Some Opinions say MAXFRAME = 1!
20.	Link Max tries	7	Enough is Enough!
21.	Link T2 timeout	100	
22.	Link T3 Timeout	18000	
23.	AX.25 digipeating	0	NOT ON BACKBONE! - OK on User Channel.
24.	Validate Callsigns	1	Keep 'em Honest!
25.	Station ID beacons	1	Keeps the MSYS/KANODE JHeard lists happy.
26.	CQ broadcasts	0	NOT ON BACKBONE! - OK on User Channel.

TXD = 500 ms on ALL Synthesized Radios - It takes TIME for TX AND RX PLL's to Lock!

SUGGESTED PARAMETER VALUES
FIGURE 3

The "Cloverleaf" Performance-Oriented HF Data Communication System

By Raymond C. Petit, W7GHM
PO Box 51
Oak Harbor, WA 98277

News Item: In March of 1990, AKØX and W7GHM conducted a series of on-the-air tests of a new HF data communication system design over the 1500-mile path between Boulder, Colorado, and Oak Harbor, Washington. On five different days, and on the 80, 40, 30, and 15-meter bands, Ed sent text from Isaiah 55 to Ray, W7GHM, at higher speeds than any other data mode was capable in the same band conditions, and Ed's "Cloverleaf" signal was so compact that twenty of them could have been packed, without mutual interference, into the same 2-kHz space now used by one packet channel. On 15 meters, Ray was printing data at 75 bit/s free of errors. On 30 and 40 meters, during a time when the packet link between AKØX and W7GHM was nothing but retries, Ray got 50 bit/s from the Clover link. (The design limit of AMTOR is 33.) On 80 meters, when it was necessary to repeat single letters several times on CW to be understood, the Clover link delivered 15 bit/s, free of errors.

During a second series of tests by Willard, N7JTQ, and Ray, W7GHM, a one-way Clover link delivered throughput-per-bandwidth performance from about 50 to over 1000 times better than HF packet in the severe multipath environment of a night-time 50-mile path on 80 meters.

The first test began at 10 PM on April 29. The Clover link transferred data at 37 bit/s on a sustained basis without losing or garbling any data. Willard and Ray were unable to establish a packet link because the TNCs retried out making connect requests.

The second test began at 9 PM on April 30, 1990. The Clover link delivered 75 bit/s with one in ten of the data blocks lost. (The two-way Clover protocol will provide for retransmission of lost blocks without requiring retransmission of blocks already correctly received.) Shortly after 10 PM, a file transfer on packet was aborted after a few minutes by a link failure. The average data rate was 0.7 bit/s. A second attempt produced a nearly identical result. The Clover one-way link then delivered 117 blocks of data at 50 bit/s, losing only two blocks.

On May 2 at 7 PM the third test was conducted. The conditions were much more stable, and a 2 kbyte file transfer on packet averaged about 31 bit/s. Afterwards, the Clover link delivered 75 bit/s. At about 8 PM, another packet file transfer averaged about 17 bit/s and it was followed by a Clover data transfer at 50 bit/s.

In the final series of tests on May 31, Logan, KL7EKI, sent the entire test message at 95 bit/s from Wasilla, Alaska, in ideal 20-meter conditions. His 9-watt signal was received 2000 miles away without a single error requiring correction. In late evening on 30 meters, when Logan and Ray's packet link was immobilized, Logan sent Clover data at 15 bits/s and 15 lines of text were received before the first uncorrectable error occurred.

The performance "figure of merit" was obtained by dividing the number of correctly received bits per second by the spacing (in hertz) required to guarantee that two signals of the same type do not cause mutual interference, even if one is 60 dB stronger than the other. For a Clover signal, this spacing is 100 Hz. Packet and AMTOR require at least 20 times this space.

Introduction

"Cloverleaf" is the name I have given to a modulation and coding scheme which offers very worthwhile improvements over HF packet, AMTOR, and even CW. The specification for this system will be a gift to the Amateur community. Here are its main performance features:

1. It is exquisitely compact in bandwidth. For practical purposes a Clover signal is entirely contained in a channel only 100 Hz wide. Clover signals are designed for channel spacing of 100 Hz; they need no guard bands. The actual channel width of the Clover receiver is this same 100 Hz. A Clover signal in the neighbor channel will not cause interference even if it is 60 dB stronger than the signal to which the receiver is tuned. A network of 10 Clover links can be maintained without mutual interference in a 1-kHz band. Each channel is a "clear channel": no collisions, no "splatter," and no key clicks come from the other users, even if they are much stronger.

2. A Clover link communicates data at the highest speed the RF propagation path permits. As the band conditions change the system adapts to them automatically. Under the best conditions, it operates at above 100 correct data bits per second delivered to the user. Under the worst conditions, conditions in which even CW data rates approach zero, a Clover link can communicate a few bits per second.

3. The Clover design uses Reed-Solomon error-control coding to correct errors in transmission, rather than rejecting a block of data on account of the errors. The coding is set such that it will recover blocks which have as many as 10% of their data symbols lost. If too many errors occur, the receiving station obtains a retransmission from the sender. It is never necessary to retransmit a block of data which has been received successfully on account of errors in previous blocks.

4. The data link is completely transparent to the data user. No restriction exists on the alphabets or data sequences. There are no illegal data codes. Input is accepted as a sequence of bytes and delivered to the output of the link in the same format. Programs using a Clover link are

totally free to define their data in the ways best suited to their needs.

5. The Clover system requires and takes advantage of the extreme frequency precision and stability of its transceivers. It also requires accurate knowledge of time. The transceivers obtain both the frequency and time information automatically from observations of WWV (for Western Hemisphere applications). A Clover channel can be centered 50 Hz from a band edge with confidence.

Overview of the Design Specification

The Cloverleaf system uses channels 100 Hz wide with their edges at multiples of 100 Hz. The Cloverleaf modulator generates 25 pulses per second in blocks which vary in length from 3 seconds to about 30 seconds. The pulses are very carefully shaped in amplitude such that they produce no sidelobes in frequency. The demodulator has dynamic range high enough to guarantee that adjacent Cloverleaf signals do not interfere with each other even if one is 60 dB stronger than the other.

For the higher speed modes, the data is transferred via the difference in the phase and amplitude of the successive pulses. Depending on the stability of the radio channel, each pulse conveys from 1 to 6 bits of data using from two to sixteen distinct phase levels and up to four amplitude levels. When the channel conditions are too poor to support the "phase" modes, the data is sent in interleaved binary pulse-position modulation to make possible, by signal averaging, the recovery of data at low speed when the signal is below the noise level.

The data is encoded into Reed-Solomon error-correcting codes which permit the receiver to recover all the data in a block—most of the time—despite the inevitable sudden losses of signal due to multipath effects and noise. The number of phase levels and the Reed-Solomon code block size used are adjusted automatically according to the channel conditions such that loss of blocks at the receiver is kept to about 10% of the blocks.

The Original Cloverleaf System

In May, 1988, I ran a simulation which satisfied me that the system described here was feasible. On June 19, 1988, I conducted an on-the-air test of a breadboard Cloverleaf modem with W7HHU on 80 meters, and it confirmed my expectations. The remainder of 1988 was spent in developing the operating system software for a 6809-based demodulator and primitive error-control coder. On January 7, 1989, AKØX successfully sent me test patterns on 15 meters from Boulder, Colorado. I spent the first half of 1989 developing an automatic synchronizing protocol and a very flexible Reed-Solomon encoder and decoder. In May, 80-meter tests with W7ZEG at a time of intense solar activity showed that there are times when nothing gets through! I spent the second half of 1989 working on the hardware for the second version.

The original breadboard version has separate transmitter and receiver modules. Both are implemented with 6809 processor boards and hand-wired expansion boards, and the software provides one-way data transfer only, for the present. The transmitter generates the pulses in software as a succession of 16-bit data words and presents them to a 16-bit D/A converter and filter. The output of the converter/filter is centered at 500 Hz and sounds like a rapid, smooth and steady hooting.

The receiver downconverts its audio input to a pair of baseband channels in quadrature, passes both signals through a 10-pole linear-phase active low-pass filter, then through digitally gain-controlled dc amplifiers, and finally to 8-bit A/D converters. With this arrangement, AGC range is about 90 dB, but distortion in the audio output stage of the radio makes only about 60 dB of it usable. The audio signal used for the downconversion is synthesized in 0.1-hertz steps and the receiver software can continually adjust the frequency over a small range to maintain zero beat.

The receiver software includes an analyzing routine which can distinguish between normal background noise and this noise with a weak carrier present. The synchronizing protocol first presents a steady carrier. This "wakes up" the demodulator which the tunes itself to zero beat. Then a stream of phase-reversing pulses establishes "framing" of pulses. This is followed by blocks of 9 pulses with interblock gaps of one pulse length. With these blocks the receiver obtains "byte synchronization." Then follows information about the modulation and coding formats in the upcoming data blocks, and a "countdown" anticipating the first block of communication data. An enormous amount of redundancy is used in this process to ensure that the synchronizing data is correctly received even under marginal conditions.

The data, text from Isaiah 55, is sent in Reed-Solomon coded blocks in the selected format. At the end of each block is a gap of one pulse length. The receiver measures the power level in this gap and the signal power level to estimate received signal-to-noise ratio. It also measures the phase jitter in the received signal and the number of errors it was required to correct in the Reed-Solomon decoder. All this information is used to keep a running estimate of the channel capacity. In a two-way protocol yet to be implemented this information would be used to request updates in the modulation and coding format employed by the remote station. The time and frequency discipline maintained by the protocol will shorten the process of establishing a connection to a few seconds.

The Second-Generation Transceivers

A pair of Clover transceivers is in development. These units will provide means for extended evaluation of this design as a practical alternative to packet under conditions requiring the highest levels of performance. Waveform generation, channel filtering and demodulation will be handled by a Motorola DSP56001 digital signal processing chip with 16-bit A/D and D/A converters. Data coding and the two-way protocol will be managed by a 6809 chip. All the system clocks are phase locked to a single master reference oscillator which is checked against WWV automatically to obtain both frequency and time synchronization. The RF portion of the transceivers feature an ultra-low noise frequency-synthesized local oscillator system which will provide reliable (and required) frequency accuracy to within a fraction of a hertz. The IF crystal filters are 1 kHz wide, permitting observation and perhaps operation on 10 Clover channels simultaneously. The linear RF power amplifiers will deliver 20 watts. An RS-232 serial port will provide the data link to the host application and computer.

INTERVIEW: Questions and Answers

Question: Why have you chosen to use PSK instead of the universally accepted narrow-band FSK?
Answer: The modulation scheme here can't really be called PSK as that term is used in amateur applications. PSK, like FSK, is supposed to be a constant-amplitude modulation method. To rapidly change the phase of a carrier is to generate enormous out-of-band radiation, and that is precisely what I wanted to avoid. FSK is better in this respect, and that is one reason it makes sense to use it on HF. But it is even better not to change the phase at all when the carrier level is nonzero. This modulator never shifts the phase of a carrier or a pulse, but generates separate smoothly amplitude contoured pulses, each of them appearing at a new phase angle as required by the data being sent. This concept is not at all new, as any communication-systems engineer can attest. But it has not been used on HF before to my knowledge, at least in the ham bands. In fact, it would have been illegal until just recently!

Question: So a lot of the motivation for this design was to make a data signal which was just as compact as possible, right?
Answer: Yes. A lot of people during the eighties worked hard on this problem and their work has now been canonized in the engineering textbooks. They were usually thinking of how to make high-speed microwave data links more efficient, but the same principles apply for low-speed systems, too.

Question: Why do you regard this extreme spectrum compactness to be so important?
Answer: What would you give for a tenfold increase in the size of the CW and data portions of every ham band? Imagine the tiny 30-meter band being expanded to the size of the entire 80-meter band! What a thought! The Clover design will accomplish the same effect, not by expanding the space, but by allowing us to pack ourselves ten times closer than we do now without mutual interference. And the specification for this system puts real meaning to the idea of "no interference." A Clover signal 60 dB stronger than the one you are listening to and only 100 Hz away won't degrade your copy. How far away must such a signal be now in any of our data modes to get the same protection? What this all amounts to is simply making better use of existing resources.

Question: Are you sure that phase-encoded data methods will work on HF? After all, a lot of people have done a lot of work, and they always have ended up using FSK.
Answer: Yes, very true. All the classic work which gave us what we have now, as far as modulation methods are concerned, was done before microprocessors existed. And before chips which were designed specifically for digital signal processing. And before ultra-stable HF radios were practical and inexpensive enough for the home builder. It's time for a reevaluation of the conventional reasoning on this subject.

Question: But what about the fundamental problem of multipath, selective fading, and Doppler shifts? Phase data is certainly going to be lost, no matter how sophisticated the hardware.
Answer: Yes. A great deal of my work with the original Clover machine was devoted to on-the-air observation. One of the neat things the Clover demodulator permits me to do is to watch the phase of any HF signal I could tune to with my ICOM 735. I began by extended observations of the 10-MHz carrier of WWV in Boulder, Colorado, about 1500 miles from here. I wrote a program which could track the carrier on a long term basis while I observed the short-term phase behavior by watching a dot move around in a circle on the scope. I also wrote a program which could display the phase spectrum of the signal on my computer screen. An another program could count the number of bit errors which would be caused by these phase shifts. There were times when WWV's phase really jumped around, but much of the time it was very stable.

Then I listened, with the same setup, to many different foreign broadcast stations. I soon discovered that some had very unstable carriers! Some were using frequency synthesizers which had residual low-frequency phase modulation in their outputs. (That was detectable because their carrier amplitude was quite constant but their phase went jiggling around wildly—phase variations caused by propagation are always accompanied by amplitude variations.) I observed many amateur CW signals and I could distinguish the transmitters using keyed oscillators from the ones using unkeyed oscillators. This all gave me a feel for what to expect.

The textbooks broadly agreed that the multipath dispersion on moderate-distance HF channels is almost always under 4 milliseconds. The degrading effect of this dispersion is reduced if the pulse duration is much greater than that figure. At bandwidths I was considering, selective fading doesn't exist; when fading occurs, virtually the entire signal fades in unison. I decided to use a 40 millisecond pulse duration, 25 pulses per second, and chose four phase levels. This would put out 50 bit/s. After a lot of analysis and experimenting with computerized Fourier analysis of signal waveforms, I found a shape that was compact with no sidelobes and was practical to generate. I built the receiving filter to match it and verified that I could recover the clock and data information from the signal after it emerged from that filter. Now to avoid errors at 25 bit/s and four phase levels in the modulation, the propagation-induced phase error during one bit intervals cannot exceed one-eighth of a cycle in 40 milliseconds, which amounts to an instantaneous frequency deviation of plus or minus just above 3 hertz. At times of intense solar activity or over some very long paths I saw that figure exceeded: I saw phase scattering which I knew would make Clover signals unreadable. But I also saw signals which were so well behaved that I decided it was worthwhile to make the modulator go to as high as 16 phase levels and to make the number of leveis adjustable to get as high a throughput as the channel permitted. So I wrote a program which could appraise the phase spectrum and recommend what format would get me the best data speed at acceptable error rates.

Concurrently with all this was the question of error checking. One of the things that really got me going on this was a statement I read in one of my textbooks to the effect that while in VHF or microwave channels the improvement due to error-control coding was at most a few dB, over HF channels the improvement could be spectacular. This really whetted my appetite! So I decided to implement a Reed-Solomon coding system, a task which took me many, many months, because I knew nothing about the math of these kind of things. My first coder operated on data blocks of 15 4-bit symbols, a block length of 60 bits. I soon discovered that a whole family of RS code block lengths would be needed in order to cope with the widely varying conditions I observed. In due course, I found out how to match the coder to the conditions.

The on-the-air tests verified my expectations. It was really beautiful to see data coming through "solid copy" even though the signal regularly dropped out or was obliterated for a moment by noise! When conditions were poor, we just hunkered down and got the data through at a slower rate. When conditions were good, we went faster! Under nearly perfect conditions it delivered 95 bit/s. If the data is coded in 5-level Baudot, we're talking about 18 characters per second. No one I know can type that fast! By comparison, AMTOR gives you about 7 Baudot characters per second maximum by design. A packet HF BBS under ideal conditions on a 200-mile path on 80-meter daytime

ground-wave conditions averaged about 10 ASCII characters per second on one occasion while sending me long messages. That was exceptional.

Question: What about the weak-signal performance?
Answer: A 10-dB signal-to-noise ratio at the receiver is perfectly adequate for data modes, but most of the time on HF we operate way, way above that level! When Ed, AKØX, and I began the 15-meter test, he was coming in something like S-8. It was way too strong. I asked him to reduce his power to as low as he could get it. I was still copying him fine when his power was so low he could not see any indication on his RF output meter! Clover, like AMTOR, works really good when the signal is weak. In the biphase mode, it can read signals which I can barely hear in the noise.

Question: How about two or more Clover signals on the same channel?
Answer: A carrier or other Clover signal 20 dB below the one you want to hear will begin to slow things down if, without it, the system is running at top speed. One of the rules the transceivers will follow is not to transmit any channel on which another Clover signal is readable. There will never be any need to, at least for a long, long time!

Question: It appears that you are turning your back on the TDMA scheme used by packet in favor of FDM . . .
Answer: Yes. Everyone knows the AX.25 protocol is really best for higher-speed work at VHF and above. Bandwidths are wide and amateur data sources, so far, are very slow in comparison. It makes good sense to "time share" a channel since any given QSO needs only a fraction of the available channel time to pass data. On HF, we don't have the bandwidth (except on 10 meters) for things like 1200 bauds on voice band FM. And wideband high-speed data modes just can't take the beating from HF propagation. So I say, instead of forcing 20 stations to coexist on one voice channel, give each one a clear channel in that same space. No collisions, no QRM! And if it really comes to push and shove, within each of these channels it is possible to establish a discipline in time, like the way CW or voice nets operate, even if the stations aren't talking to each other.

The Version 2 Transceiver

Question: Have you considered multitone modulation formats?
Answer: Yes, quite a bit, and on several occasions. My IF filters on the Version 2 system are 1 kHz wide, centered at a point 2.5 kHz below the BFO. The DSP chip can easily generate more than one tone. But unless the linear amplifier intermod products are kept down, those products will interfere with signals in the nearby channels. Several times I pondered that some good power FETs are specified at −50 dB third-order products in class A operation at about 50 watts. That's not bad. This is a promising area.

Question: Will the Version 2 transceivers be capable of operating on the other data modes?
Answer: The heart of the transceiver is a Motorola DSP56001 digital signal processing chip. A 6809 handles everything else. The transceivers should be able to do anything. For the present, I'll stick to Clover formats, but the hardware of the transceivers is very "general purpose," and I'd invite any enterprising programmers to make this matching jump through whatever loops they please.

Question: What about the RF hardware for these transceivers? The frequency synthesizers look a bit complicated for a home builder.
Answer: These transceivers are designed for just one purpose: high-performance HF data communication. They have no extras, no bells and whistles. The RF paths are incredibly simple in comparison to your typical commercial rig. It is single conversion, and the IF frequency, 18 MHz minus 2.5 kHz, works with the 20-30 MHz synthesizer to provide coverage from about 2 to 12 MHz. They were designed with frequency precision and stability as a primary goal. The IF amplifier has no AGC, but instead is designed for high dynamic range. The DSP engine gets its data from a 16-bit A/D converter—that provides 90 dB range—and all the channel filtering is done in software.

More than half of the circuitry is in the frequency synthesizer. It is a multiloop design which will provide one-tenth hertz steps over its entire 10-MHz range. Four of the five circuit boards of the synthesizer are identical. Most of the parts for the transceiver were obtained from a well-known mail-order house in the Midwest (USA). The crystal filter is made from inexpensive microprocessor clock crystals. The linear amplifiers are straight out of a Motorola application note. I think any experienced home builder could do it fine.

Question: It still looks pretty expensive, out of amateur range . . .
Answer: Yes it is, but in ten years one of these transceivers will be possible to build at very attractive prices. My versions are synthesized for flexibility, but in the applications I see for these things, a single-frequency crystal-controlled version at half the cost makes good sense. And just watch the prices for DSP chips fall as they are more widely used!

Question: Where do you see this system fitting into the whole of amateur data communication?
Answer: My experience with Coherent CW says that I shouldn't expect something like this to gain wide acceptance quickly. AMTOR showed what could be done when someone was designing specifically for the conditions we get on HF. Packet is gaining a wide following in spite of its performance on HF. I see that some people are thinking about making incremental improvements to HF packet: It's needed and all to the good. But at some point, a whole new approach has to be taken. It's like the conversion from am voice to single sideband back in the 50s.

I expect that the Cloverleaf system will find its best use in the lower-frequency HF bands where it is well suited to the formation of regional networks covering areas the size of Western Europe. Link-level protocols are being worked out, but networking software is a project awaiting a taker. I doubt if Clover will replace packet, but it will provide a high-performance alternative. Getting twice the throughput in one-twentieth the bandwidth at some point just has to make sense!

Question: Your introduction indicated that you intended to give this whole system to the amateur community. Haven't you considered patenting this thing and making a commercial enterprise of it?
Answer: Yes, I've thought about it a lot. Back in 1972, I made a commercial enterprise out of a fun project and I survived for about 3 years. I'm a very small budget operation, your proverbial garage outfit. Getting a patent costs more than my entire annual budget, it takes several years, no protection exists until you have it, and then in order to protect it you have to spend hundreds of thousands and all your emotional energy in the courts. I don't want to live that way. It's more important to me that the machine gets used, that practical Clover networks get established, and that we all get to enjoy its advantages. I expect to have diagrams, circuit boards, and parts kits for home builders pretty soon. I expect that eventually some companies will start making these things. I plan to be one of them.

FREQUENCY-STABLE NARROWBAND TRANSCEIVER FOR 10100.5 KHZ

Raymond C. Petit, W7GHM

The accompanying diagrams will permit the experienced home builder to assemble a transceiver suitable for coherent-CW and Clover operation on the bottom end of the 30 Meter ham band. It has been designed for high dynamic range (SL6440C high-level mixers), exceptional i.f. selectivity (9-pole 300 Hz filters), and ultra-stable operation (LO and BFO phaselocked to a frequency standard). It is kept as simple as possible and the frequency plan makes use of inexpensive microprocessor clock crystals. It is also designed with thought to the future, when its single-frequency stabilized HFO can be replaced with an 18-19 MHz frequency synthesizer.

I recommend building this unit as a set of shielded modules connected by r.f. cables. Bring the power supply lines through the shielded boxes with feedthrough capacitors. The CMOS-level signals can be routed via SHORT UNTERMINATED 50 ohm cables. All the signal path and oscillator r.f is matched to 50 ohms. This makes it easy to build and check out each module separately.

The transmitter and receiver sections have separate but identical crystal filtes and front-end filters. RF switching of a single crystal filter and a single front-end filter can be used instead, but dual-filter arrangement is no more expensive. The filters are based on the Cohn design described by Hayward in QST (July 1987) with a slight variation: The center frequency can be set over a small range by choice of the "offset" capacitors (C10-C18 in the filter schematic).

Unlike most IF amplifiers, this one operates at fixed gain--no agc. The combined gain of the IF amplifier and product detector is about 50 dB. The audio output varies from about 30 microvolts to about 1 volt, a 90 dB range. The Clover TNC does the AGC compensation in its audio processor.

The input and output filters have a bandwidth of about 1 MHz, about 2 dB insertion loss, and very steep skirts: the input mixer on the receiver is very well protected from strong out-of-band signals! To align the input filter, connect a 10.5-MHz sinewave r.f. generator with 50 ohm source resistance to one of its ports, and a scope probe to THE SAME POINT. Terminate the other port with 50 ohms. Set all the trimmer capacitors to minimim capacitance. Then set the trimmer closest to the source and probe for MINIMUM r.f. voltage, and do not change it afterward. Then set the next trimmer for MAXIMUM r.f voltage, and don't change it afterward. Set the third trimmer for MINIMUM, the fourth for MAXIMUM, and the fifth for MINIMUM without changing previous settings. It's done: five adjustments, no tweaking!

The 16-MHz oscillator provides a signal required by the (future) frequency synthesizer, the 8 MHz BFO signal, and the clock signal required by the Clover TNC. The synthesizer and BFO outputs are

0 dBm /50 ohm sinewaves. Two separate switchable BFO outputs are provided for T/R switching purposes. With R10 removed it should be possible to get the oscillator to deliver 16 MHz exactly for some d.c. voltage between 2.0 and 2.5 volts applied to the junction of C13 and R9. L5 can be changed if necessary to get it on frequency. With R10 in place, the frequency standard connected to J3 and the jumper set to correspond to the frequency of the standard, correct operation will be indicated by a clean rectangular waveform at U3 pin 6. This waveform will change its duty cycle slightly if the crystal is heated, but the frequency of the oscillator will stay exactly at 16 MHz.

The 2.1 MHz LO circuit uses the same basic scheme for the oscillator and phase detector. The divide-by 21 counter functions by alternately counting out 10 input pulses and then 11 pulses. Extensive decoupling and division by 4 at the output are intended to prevent switching noise from getting into the 2.1 MHz output.

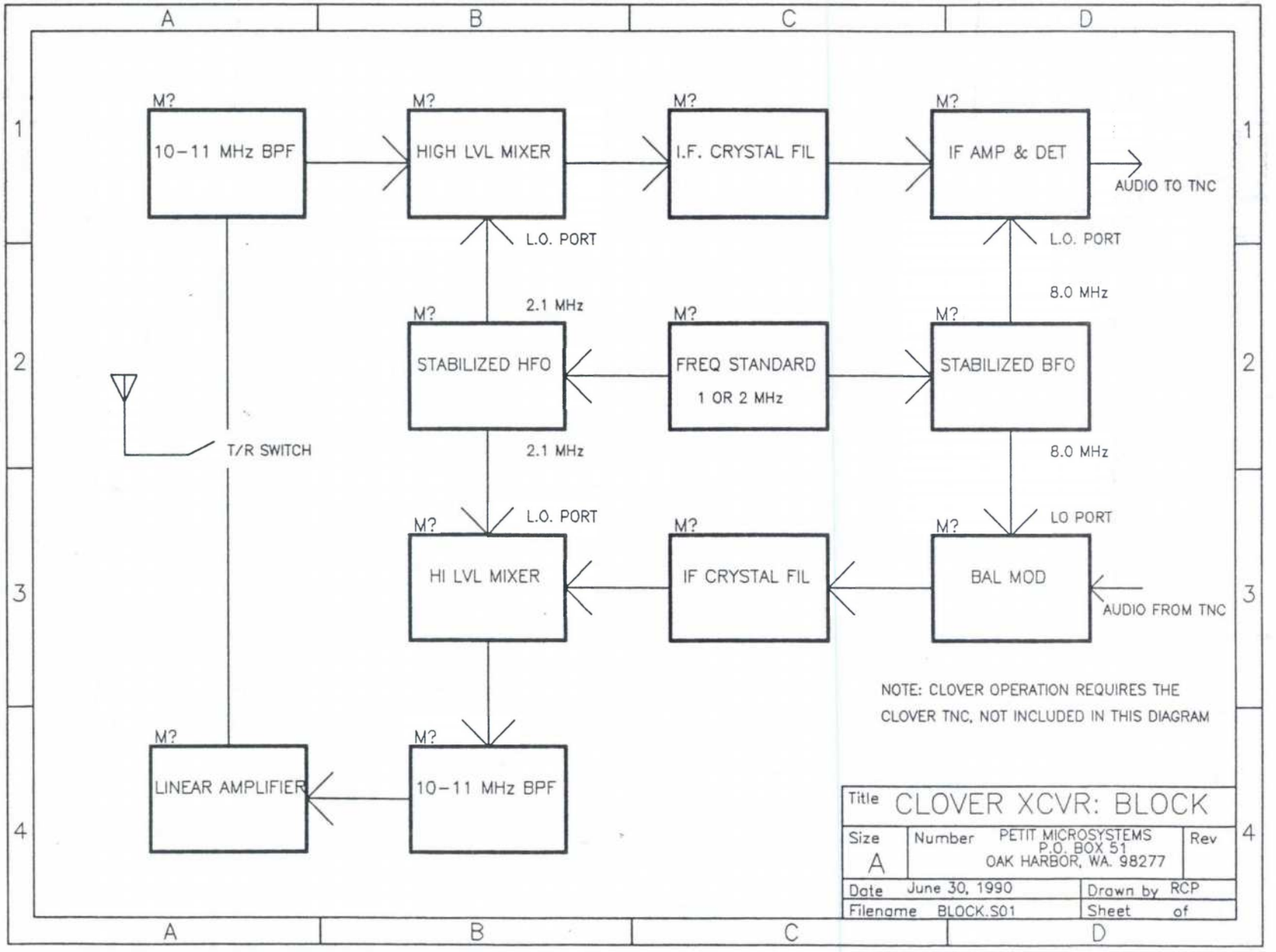
The RF output of the high-level mixer on the transmitter side should be set for about 10 dBm (1 volt peak/50 ohms). Broadband Class A drivers delivering from 0.2 watts to about 2 watts are shown in the ARRL Handbook. Linear amplifiers can be built from plans and parts kits supplied by Communication Concepts, Inc., 508 Millstone Drive, Xenia, Ohio 45385 (513-426-8600).

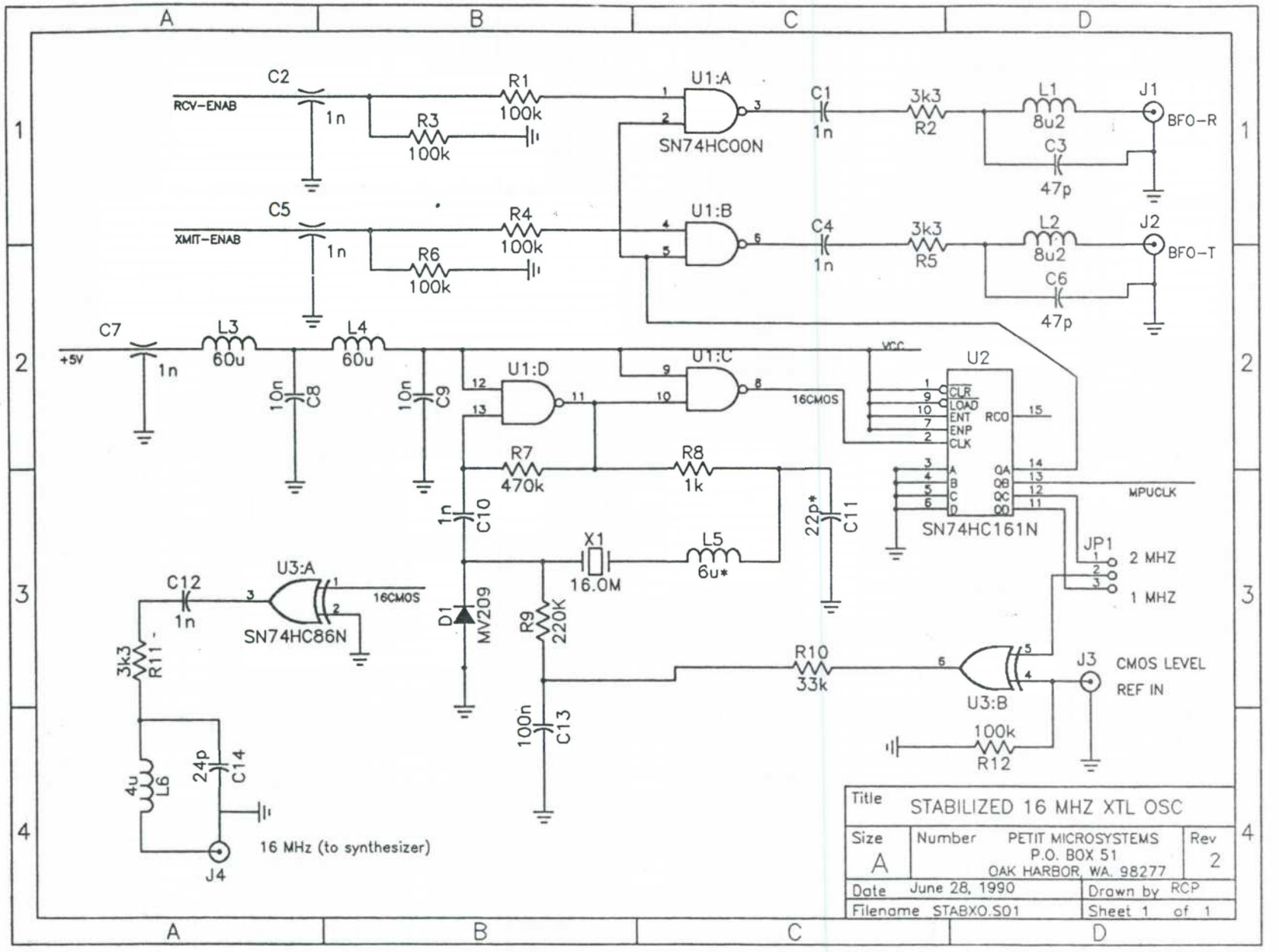
An oven-stabilized 2-MHz frequency standard meeting the requirements of Clover operation can be obtained for $12.95 plus shipping from Fair Radio Sales, P.O. Box 1105, Lima, Ohio, 45802 (419-223-2196).

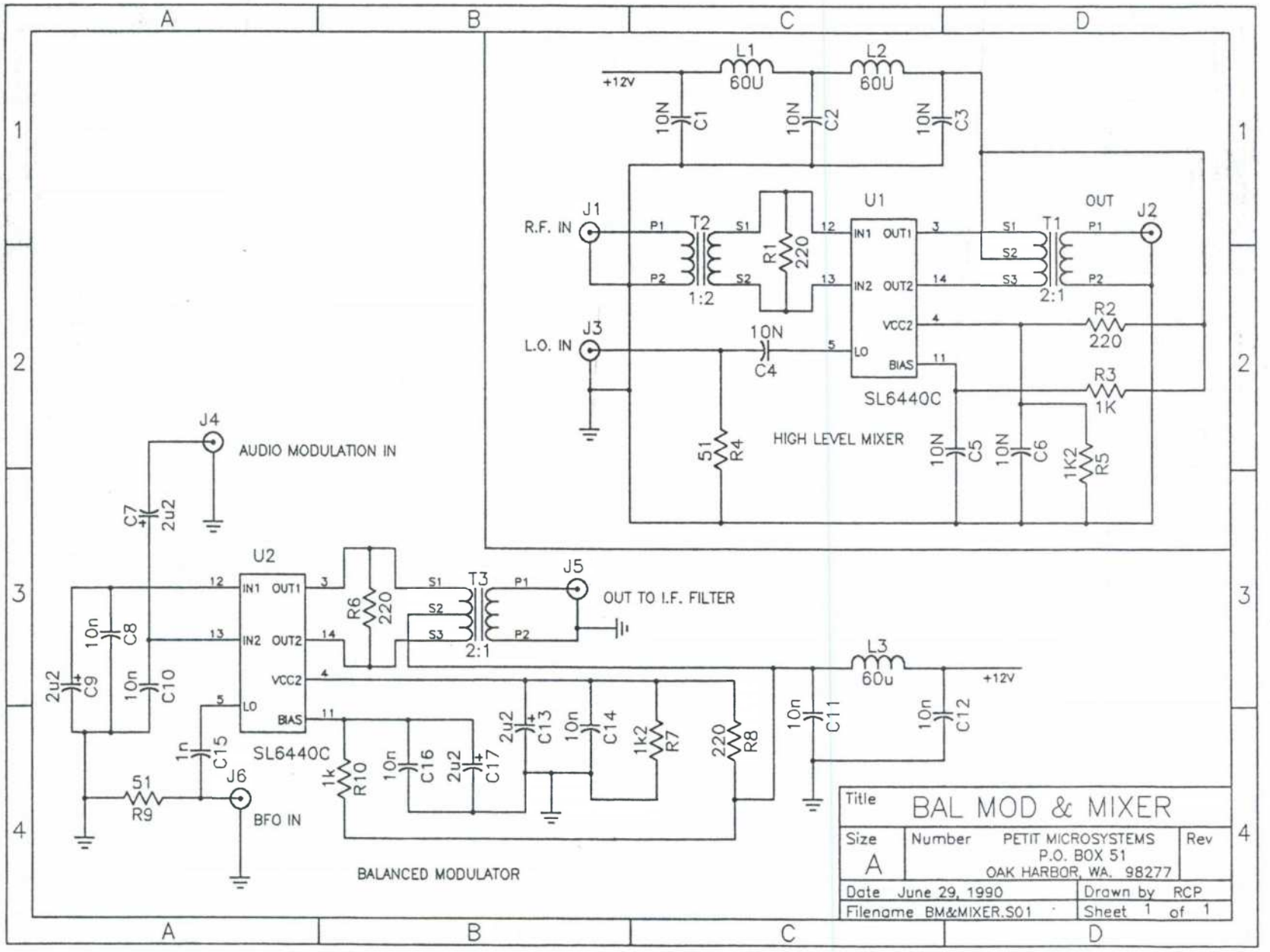
Most of the parts needed for this project can be obtained from Digi-Key Corporation, P.O. Box 677, Thief River Falls, MN 56701 (800-344-4539). I have a good stock of the SL6440C mixers, ferrite beads for the 60 uH r.f. chokes and r.f. transformers, and toroid cores for L1-L5 in each of the two front-end filters. If there is sufficient interest, I can supply a complete parts kit including circuit boards as a commercial product.

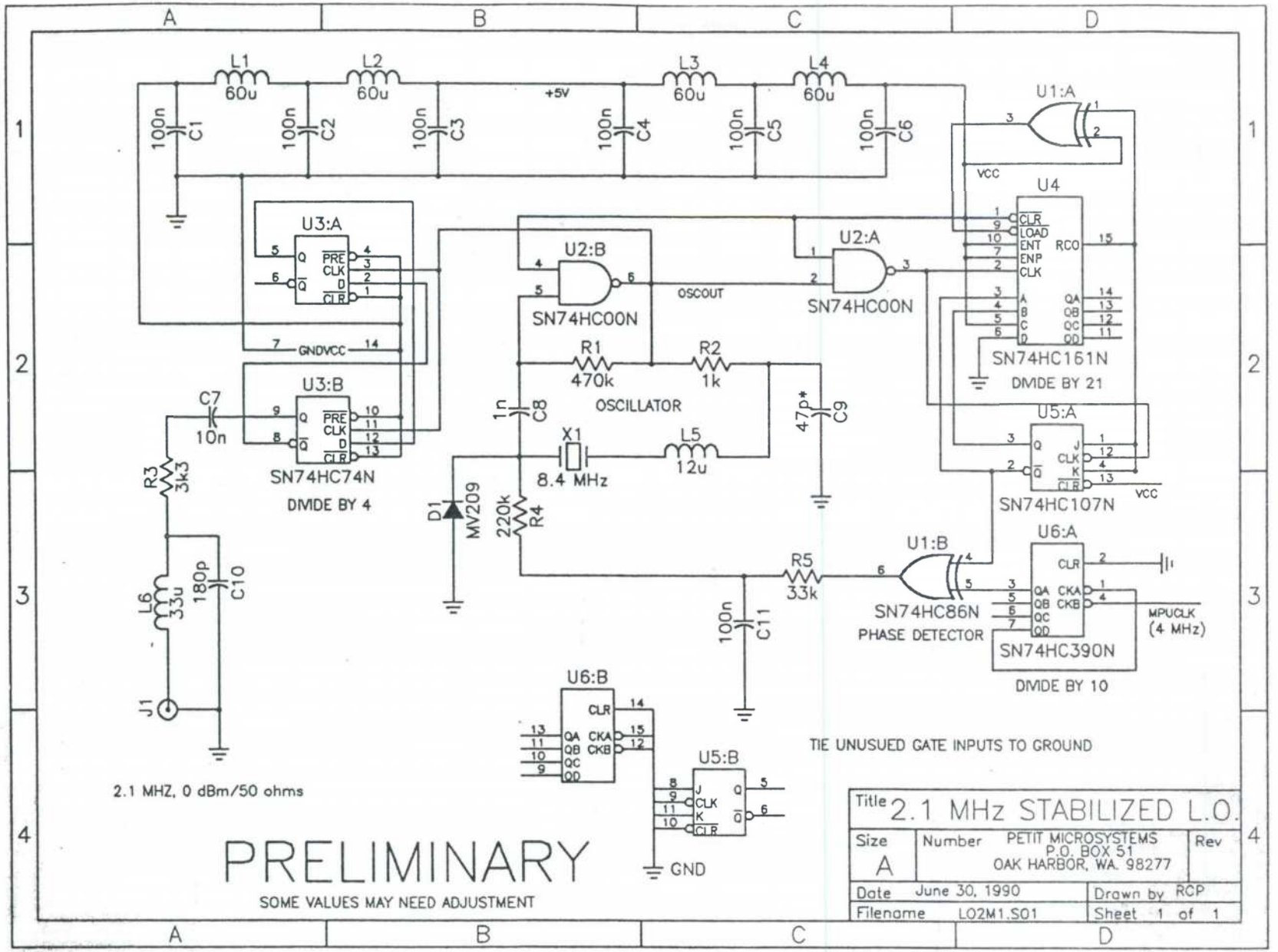
I hope the FIRST CLOVER NETWORK can get on the air before the end of the year, using some of these transceivers. UP FRONT, I inform you that you will need FOUR items for Clover operation: The frequency standard, the transceiver (with at least a few watt linear amplifier), a CRT terminal or a computer with communications software and an RS-232 port, and the CLOVER single-channel TNC which I plan to offer as a commercial product for about $300.

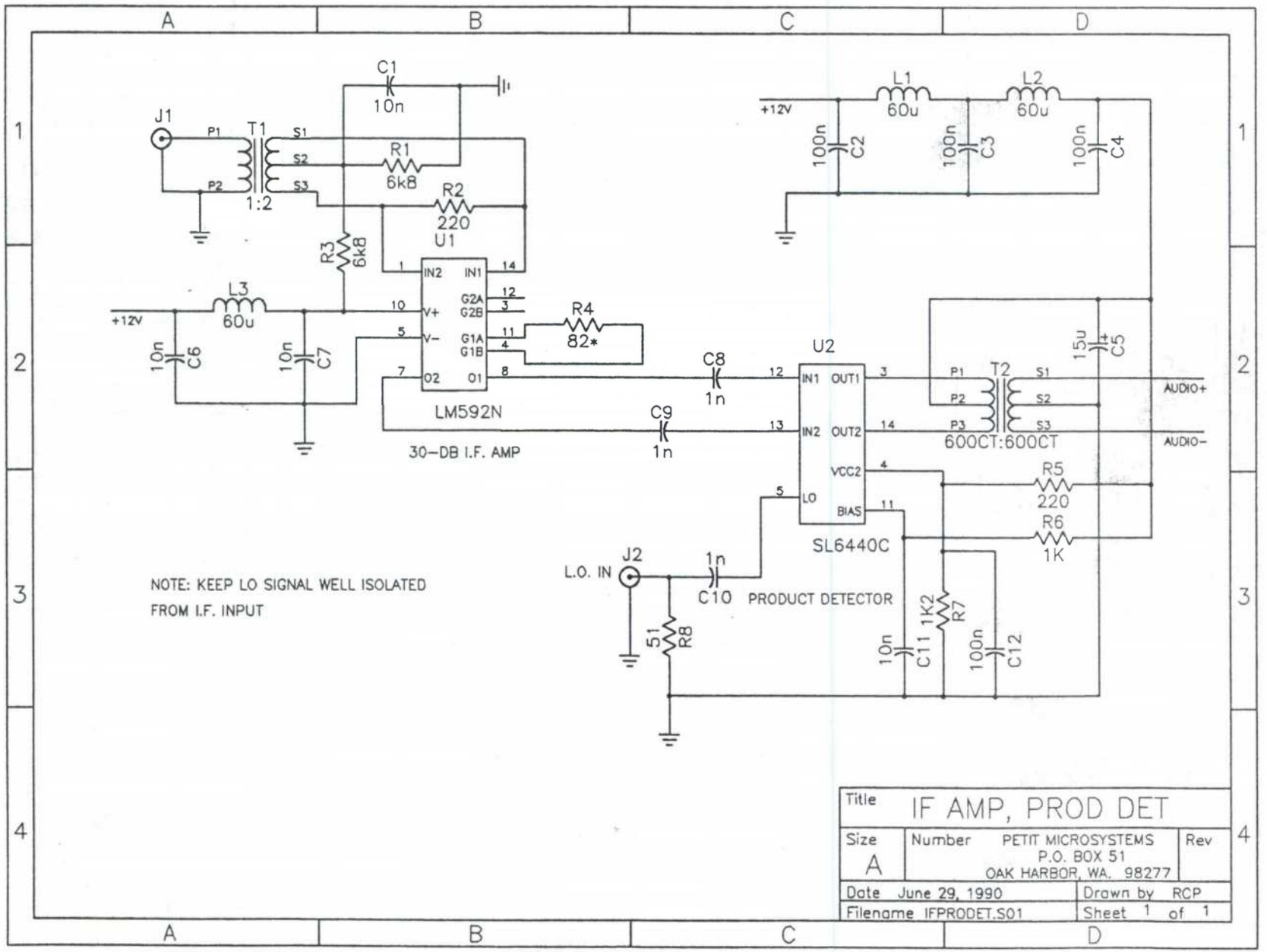
Please keep me informed of your progress as we build the first operational CLOVER systems!

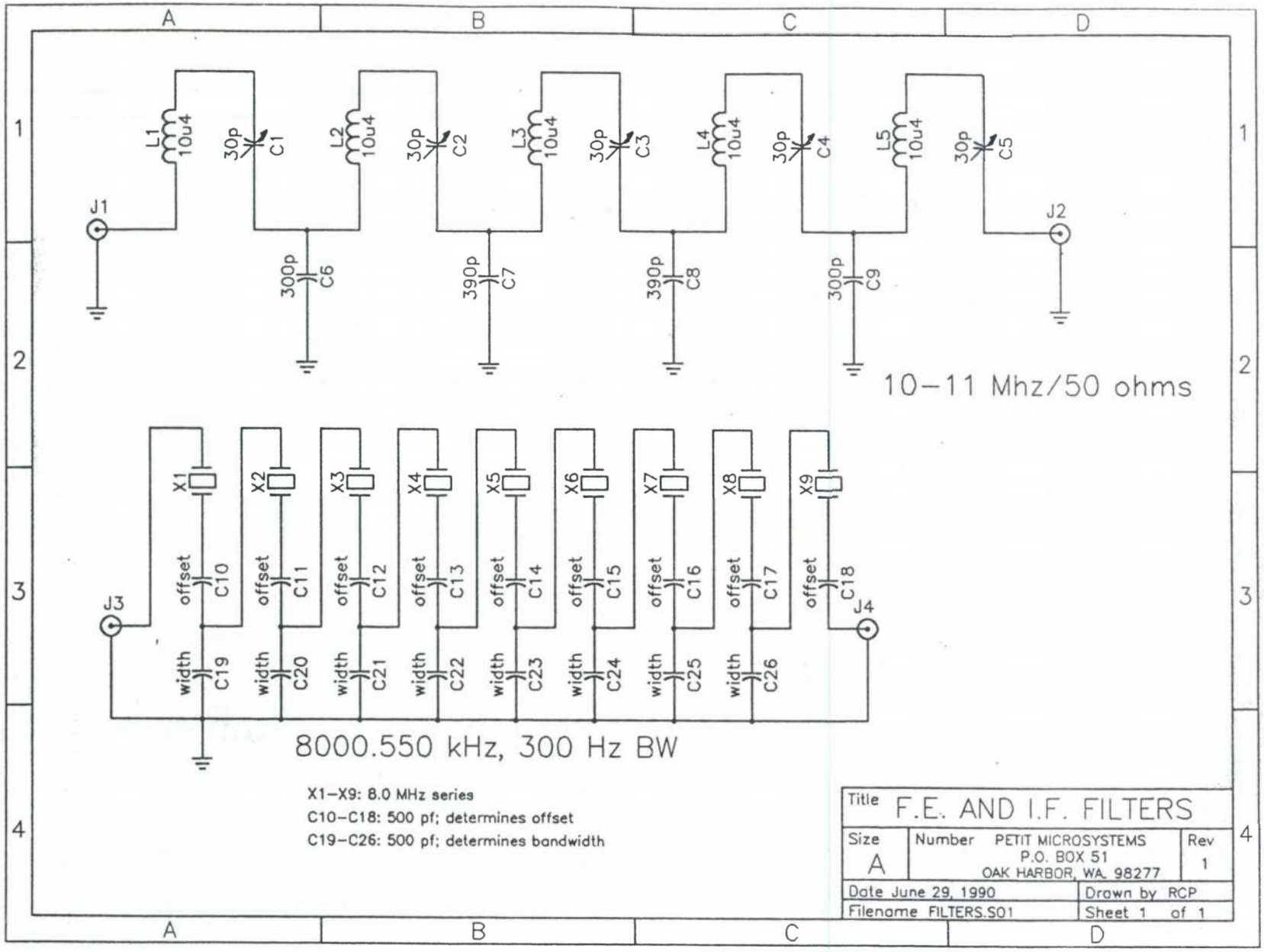

A
B
C
D
1
2
3
4
M?
10–11 MHz BPF
HIGH LVL MIXER
I.F. CRYSTAL FIL
IF AMP & DET
AUDIO TO TNC
L.O. PORT
L.O. PORT
2.1 MHz
8.0 MHz
STABILIZED HFO
FREQ STANDARD
1 OR 2 MHz
STABILIZED BFO
T/R SWITCH
2.1 MHz
8.0 MHz
L.O. PORT
LO PORT
HI LVL MIXER
IF CRYSTAL FIL
BAL MOD
AUDIO FROM TNC
NOTE: CLOVER OPERATION REQUIRES THE
CLOVER TNC, NOT INCLUDED IN THIS DIAGRAM
LINEAR AMPLIFIER
10–11 MHz BPF
Title CLOVER XCVR: BLOCK
Size A
Number
PETIT MICROSYSTEMS
P.O. BOX 51
OAK HARBOR, WA. 98277
Rev
Date June 30, 1990
Drawn by RCP
Filename BLOCK.S01
Sheet of


A
B
C
D
1
2
3
4
RCV-ENAB
XMIT-ENAB
C2
1n
R1
100k
R3
100k
C5
1n
R4
100k
R6
100k
U1:A
SN74HC00N
U1:B
C1
1n
C4
1n
3k3
R2
3k3
R5
L1
8u2
L2
8u2
C3
47p
C6
47p
J1
BFO-R
J2
BFO-T
C7
1n
+5V
L3
60u
L4
60u
10n
C8
10n
C9
U1:D
U1:C
16CMOS
VCC
U2
SN74HC161N
CLR
LOAD
ENT
ENP
CLK
RCO
QA
QB
QC
QD
MPUCLK
R7
470k
R8
1k
1n
C10
X1
16.0M
L5
6u*
22p*
C11
JP1
2 MHZ
1 MHZ
U3:A
SN74HC86N
C12
1n
3k3
R11
D1
MV209
R9
220K
R10
33k
U3:B
J3
CMOS LEVEL
REF IN
100k
R12
100n
C13
4u
L6
24p
C14
16 MHz (to synthesizer)
J4
Title
STABILIZED 16 MHZ XTL OSC
Size
A
Number
PETIT MICROSYSTEMS
P.O. BOX 51
OAK HARBOR, WA. 98277
Rev
2
Date
June 28, 1990
Drawn by RCP
Filename STABXO.S01
Sheet 1 of 1


L1
60U
L2
60U
+12V
10N C1
10N C2
10N C3
J1
R.F. IN
T2
1:2
R1
220
U1
SL6440C
OUT
J2
T1
2:1
R2
220
J3
L.O. IN
10N
C4
R3
1K
51
R4
HIGH LEVEL MIXER
10N C5
10N C6
1K2
R5
J4
AUDIO MODULATION IN
C7
2u2
U2
R6
220
T3
2:1
J5
OUT TO I.F. FILTER
2u2 C9
10n C8
10n C10
SL6440C
L3
60u
+12V
10n C11
10n C12
1n C15
J6
BFO IN
51
R9
1k
R10
10n C16
2u2 C17
2u2 C13
10n C14
1k2 R7
220 R8
BALANCED MODULATOR
Title
BAL MOD & MIXER
Size
A
Number
PETIT MICROSYSTEMS
P.O. BOX 51
OAK HARBOR, WA. 98277
Rev
Date June 29, 1990
Drawn by RCP
Filename BM&MIXER.S01
Sheet 1 of 1


L1 60u
L2 60u
L3 60u
L4 60u
+5V
100n C1
100n C2
100n C3
100n C4
100n C5
100n C6
U1:A
VCC
U4
CLR
LOAD
ENT
ENP
CLK
RCO
A
B
C
D
QA
QB
QC
QD
SN74HC161N
DMDE BY 21
U3:A
PRE
CLK
D
CLR
Q
GNDVCC
U2:B
SN74HC00N
OSCOUT
U2:A
SN74HC00N
R1 470k
R2 1k
OSCILLATOR
1n C8
47p* C9
X1 8.4 MHz
L5 12u
U5:A
J
K
SN74HC107N
C7 10n
U3:B
SN74HC74N
DMDE BY 4
R3 3k3
D1 MV209
220k R4
R5 33k
U1:B
SN74HC86N
PHASE DETECTOR
U6:A
SN74HC390N
DMDE BY 10
MPUCLK (4 MHz)
100n C11
L6 33u
180p C10
J1
U6:B
U5:B
GND
TIE UNUSUED GATE INPUTS TO GROUND
2.1 MHZ, 0 dBm/50 ohms
PRELIMINARY
SOME VALUES MAY NEED ADJUSTMENT
Title 2.1 MHz STABILIZED L.O.
Size A
Number
PETIT MICROSYSTEMS
P.O. BOX 51
OAK HARBOR, WA. 98277
Rev
Date June 30, 1990
Drawn by RCP
Filename LO2M1.S01
Sheet 1 of 1


A
B
C
D
1
2
3
4
C1
10n
J1
P1
P2
T1
S1
S2
S3
1:2
R1
6k8
R2
220
R3
6k8
U1
IN2
IN1
G2A
G2B
V+
V-
G1A
G1B
O2
O1
LM592N
30-DB I.F. AMP
R4
82*
L3
60u
+12V
10n
C6
10n
C7
C8
1n
C9
1n
L1
60u
L2
60u
100n
C2
100n
C3
100n
C4
15u
C5
U2
IN1
OUT1
IN2
OUT2
VCC2
LO
BIAS
SL6440C
T2
P1
P2
P3
S1
S2
S3
600CT:600CT
AUDIO+
AUDIO-
R5
220
R6
1K
L.O. IN
J2
1n
C10
PRODUCT DETECTOR
51
R8
10n
C11
1K2
R7
100n
C12
NOTE: KEEP LO SIGNAL WELL ISOLATED
FROM I.F. INPUT
Title
IF AMP, PROD DET
Size
A
Number
PETIT MICROSYSTEMS
P.O. BOX 51
OAK HARBOR, WA. 98277
Rev
Date June 29, 1990
Drawn by RCP
Filename IFPRODET.S01
Sheet 1 of 1


A
B
C
D
1
2
3
4
L1 10u4
L2 10u4
L3 10u4
L4 10u4
L5 10u4
30p C1
30p C2
30p C3
30p C4
30p C5
J1
J2
300p C6
390p C7
390p C8
300p C9
10–11 Mhz/50 ohms
X1
X2
X3
X4
X5
X6
X7
X8
X9
J3
J4
offset C10
offset C11
offset C12
offset C13
offset C14
offset C15
offset C16
offset C17
offset C18
width C19
width C20
width C21
width C22
width C23
width C24
width C25
width C26
8000.550 kHz, 300 Hz BW
X1–X9: 8.0 MHz series
C10–C18: 500 pf; determines offset
C19–C26: 500 pf; determines bandwidth
Title F.E. AND I.F. FILTERS
Size A
Number PETIT MICROSYSTEMS P.O. BOX 51 OAK HARBOR, WA. 98277
Rev 1
Date June 29, 1990
Drawn by RCP
Filename FILTERS.S01
Sheet 1 of 1

PACSAT Protocol Suite - An overview

Harold E. Price, NK6K
Jeff Ward, G0/K8KA

ABSTRACT

A Low Earth Orbiting "Pacsat" has been described in the past as an orbiting bulletin board system. This is an over-simplification. A PACSAT is a multi-channel, full duplex device, with short, periodic access times dictated by orbital mechanics. These attributes mandate a different approach than the standard command-line interpreter style of BBS if the full potential of a PACSAT is to be realized.

The authors propose a new methodology for a PACSAT, and have developed several new protocols to implement more efficient access. These protocols all use AX.25, either in connected mode or with UI frames. This paper provides a description of the access model, and an overview of the new protocols.

Background

The authors have been struggling with the question "How can we make the best use of a bandwidth-limited low earth orbiting digital store-and-forward system with a worldwide, unstructured, heterogeneous user base" since an amateur Packet Radio satellite was first discussed in 1982. We began on air experimentation with the UoSAT-2 (UO-11) Digital Communications Experiment in December, 1984. In the following five and one half years, we've looked at where a resource like a PACSAT best fits in to the network as a whole. As a result of our study, we are proposing the use of a broadcast protocol as the basic downlink method, and a "file server" rather than a BBS application as the basic service offered. This document provides a brief overview of these conclusions, the companion specification documents provide the implementation details.

This paper and the companion protocol specification papers assume that the reader has a basic understanding of the current packet radio satellites, for additional background, see references [1] through [6].

PACSAT

PACSAT is generic term in the amateur radio service for a low earth orbiting spacecraft which carries a large on-board memory for the purpose of data storage and retrieval by groundstations. A PACSAT can be the entire mission of a spacecraft, such as AMSAT-NA's AO-16, or a minor adjunct, such as the DCE on UO-11. The paper refers to the current "PACSAT" spacecraft - the University of Surrey's UoSAT-3 (UO-14) and the AMSAT Microsats AO-16 and LO-19. These spacecraft will be the hosts of software developed by the authors which implements the protocols described herein.

Each of these spacecraft are different. AO-16 and LO-19 are the most closely related, based on AMSAT's Microsat design. From the user's point of view, they have four 1200 bps uplinks and one 1200 bps downlink. These are switchable to 4800 bps, but no ground modems exist at this time. UO-14 has a single uplink and downlink, at 9600 bps. Although the onboard computers are different, they are compatible at the application software level, permitting the same software to be used on all three.

In spite of these differences, all of these spacecraft share the following attribute: each is a bandwidth limited device. The number of uplinks and downlinks is much less than the number of users, and the capacity of the link is much less than the offered load. Each is only visible to a particular user for about 14 minutes, four or five times a day at middle latitudes. We feel that this is the critical

design driver, and the access methods must be optimized with this in view.

Keep in mind, however, that even while subject to access time limitations, the satellites can still move a prodigious amount of data, especially when compared to the current amateur radio long haul network standards. A typical gateway station, moving traffic from the US to the UK on 20MHz at 300 baud, assuming the band is open for 16 hours, could move 1.7 million bytes of data per day, if the link was 100% efficient. The average HF link is not 100% efficient, at best it is perhaps 30% efficient. The link is only half duplex, so this data transfer is only way only.

UO-14, even with only 56 minutes of access time per day at 9600 baud, can move 3.2 million bytes of data in one direction. The excellent link quality of the current PACSATs, combined with their full duplex nature and the protocols we are proposing, can approach 90% efficiency. Full duplex means transfers can occur in both directions simultaneously, so that UO-14 could move nearly 5.7 million bytes of data between the US and the UK in a 24 hour period, vs. .5 million bytes over an HF circuit.

The desire to realize this potential is the reason we choose some non-traditional (for the amateur radio service) access methods for PACSAT. These methods, broadcasting and file server, are discussed below.

Broadcasting

A spacecraft is inherently a broadcast device. It transmits from on high, and many users can hear it at the same time.

To optimize the available downlink time, we are recommending the use of a broadcast protocol. This protocol adds information to the basic AX.25 data frame to permit many stations to make simultaneous use of a single file download session. When one station in Maryland requests the current orbital element sets, there is no need for stations in Toronto and Miami to do the same, they should be able to make use of the information as it is downlinked to Maryland if they are all in view of the satellite at the same time. To make use of a broadcasted frame of data, each frame must be tagged with the file it belongs to and the position within that file that the data belongs in.

There should also be enough information for a station to determine if it has all of the data belonging to a file, and if not, to request that just the missing parts of the file be retransmitted. The specification titled "PACSAT Broadcast Protocol" describes a method of providing this additional information.

With a broadcast protocol, a groundstation can simply monitor the downlink and accumulate files of data. Since files gathered in this way will have been unsolicited, the format of the contents may not be known to the user. For example, if one asked for a file of NASA format orbital elements, one can make a good guess that the resulting file contains NASA format orbital elements. However, if a "random" file is captured, its contents may not be understandable simply from inspection. Some addition information, such as a file name, data type, description, creation date, etc., may be required. Each broadcasted file, therefore, needs a header in a standard format with this information. The specification titled "PACSAT File Header Definition" describes a method of providing this information.

We hope that the broadcast protocol promote efficient use of the downlink. It should reduce the number of requests for files of general interest. It should also reduce the uplink loading, since a broadcasted file does not receive an ack for each frame or group of frames. In the best case, only one "ack" is sent for an entire file, and that would be the request to stop broadcasting it.

Even though the sky-to-ground link is broadcast in nature, the ground-to-sky link is not. PACSAT "sees" many ground stations at one time. For this reason, a connected-mode, non broadcast file transfer method is also defined, and is described in the paper on "PACSAT File Transfer Level 0".

File Server

As a data transfer and storage device, a PACSAT can serve a multitude of purposes. It can store telemetry, digitized voice and video images, personal mail, forwarded mail, or anything else the can be stored in a computer file. Mail forwarding is a good example of an excellent use of a PACSAT. AO-16's 1200 baud link could easily be used to transfer 240k bytes of uncompressed forwarded mail in each direction between California and England in 24 hours, with just one morning

and evening pass over each location. UO-14's 9600 baud link could move 1.6 Mb of data in the same time. A PACSAT can store up to 8Mb of data. This would make a powerful addition to the current HF relay network.

The problem, however, is that the current amateur network is in a state of flux. New addressing schemes are proposed every few weeks, new routes and new ways of routing are proposed, tried, discarded or modified. This is good. Implementing the software on a spacecraft to follow these shifting designs is difficult, however. The testing required for the spacecraft is more rigorous, especially on the Microsats, where the same computer is used for the BBS and to keep the batteries charged. Faulty forwarding code could crash the computer, which could cause damage to the batteries or reduce their life expectancy.

The amount of program memory is limited on the spacecraft as well. To counter the effects of high energy particles above the earth's atmosphere which cause memory bits to be changed, the PACSATs use 12 bits to store 8 bits of program data. The extra bits are used to correct for single bit errors. To keep the cost down, and to reduce the power used (AO-16's CPU uses about 500 milliwatts, on average), only 256k bytes of program space is available. (This should not be confused with the message storage space, which is much larger than the program memory, and is protected with a software algorithm using three 3 bytes to protect 253 bytes of data. Because this memory is protected with software, it is not suitable for storing a running program, since a program can not protect its executing instruction.)

We have a desire, then, to keep the spacecraft code simple and stable, while still allowing it to be a useful part of the changing amateur network.

We propose that the spacecraft be primarily used as a file server, moving data files from one point to another. The PACSAT would have no knowledge of the contents of the files, nor would it take an active role in the forwarding of mail messages. Groundbased software could, however, make the PACSAT system look like a familiar BBS to the user, and it could intelligently forward mail.

A PACSAT will know how to receive and transmit a standard file format. All files will have a standard header, the same one that is used by the broadcast protocol. It will also know how to select files for transmission based on the contents of the header. This feature can then be used by groundstation software to emulate any desired user interface.

For example, assume that a user wanted to send a personal mail message to a friend. In the current terrestrial environment, he would connect to a BBS, which would lead him in a question and answer session something like this:

Remote Computer	User
What do you want?	Send message
To whom?	Fred
Title?	Club meeting
Message?	Meeting at 8 p.m.
What do you want?	Read new mail.
Message #200	
. . . . .	

Using the PACSAT system, exactly the same exchange would take place, except that the conversation is between the user and his local computer. The message is stored for later transmission to a PACSAT. The read new mail request is also stored. The next time the PACSAT comes overhead, the computer does the following:

1) builds a file with a standard PACSAT header. The header says that the file contains a mail message, from you, to Fred.

2) The file is compressed, and sent to PACSAT.

3) The local computer then sends a message to PACSAT that says "send the next file who's header meets the following criteria: it's a mail message type, the destination is me, and the file number is bigger than x".

"x" is the number of the last file received on the ground, and is kept by the local computer. After the pass, the local computer can now print any new mail received. To the user, it looked pretty much the same.

What about file forwarding? A gateway would need to know what type of mail it could forward. Let's assume that the routing scheme of the week is based on a hierarchical string containing states, like nk6k.ca.usa, and this gateway handles mail to CA, NV, and OR. The gateway would send a message to PACSAT containing the following request:

"Send the next file who's header meets the following criteria: it's a forwarded message, and the destination string contains '?.ca.?' or '?.nv.?' or '?.or.?', and the download count is 0."

The file would be received, decompressed, and imported into the standard BBS program after the pass.

In this way, the ground program can be as simple or as complex as required, the PACSAT only needs to know how to select a file for transmission based on the contents of fields in the standard file header.

Summary

These two ideas, broadcasting and file server, are certainly different than the current common usage of packet radio on the amateur bands. We feel that this is the best approach for the special case of a PACSAT, however, and that with suitable groundstation software, these concepts can be integrated into the mainstream.

Implementation Status

Prototype implementations of all of the protocols discussed in this group of papers are running on UO-14 as of late July, they should be running on the Microsats by the time of the ARRL conference. Prototype ground software is also running. We plan to make the source code for simple versions of the ground portion of the system available asap. Executable versions for the IBM PC will be made available as shareware, with the proceeds going to AMSAT-UK and AMSAT-NA to further developement of future PACSATs. Fully integrated, automated, color graphic, "all singing and dancing" software will be available for sale by AMSAT-UK and AMSAT-NA later in the year. Like QUICKTRAK and InstantTrak, the proceeds from this commerical quality software will go to finance future amateur satellite endevours.

We hope that other software authors will use the documentation and source to develop support for non IBM PC systems. The contents of these papers are sufficient to allow programmers to begin implementing their own software now.

Correspondence

The authors may be reached at:

Telemail:	HPRICE or UOSAT
Compuserve:	71635,1174
Packet:	NK6K @ WB6YMH or G0K8KA @ GB2UP
Internet:	71635.1174 @COMPUSERVE.COM
Mail:	Jeff Ward UoSAT Unit University of Surrey Guildford, Surrey GU2 5XH UK

REFERENCES

[1] Loughmiller D., and McGwier, R., "Microsat: The next Generation of OSCAR Satellites - Part 1", *QST*, May 1989, pp. 37-40.

[2] Loughmiller D., and McGwier, R., "Microsat: The next Generation of OSCAR Satellites - Part 2", *QST*, June 1989, pp. 53-54,70.

[3] Clark, T., "Amsat's Microsat/Pacsat Program", *ARRL 7th Computer Networking Conference*, pp. 41-47, Columbia, Maryland, 1 October 1988.

[4] Ward, J., "The UoSAT-D Packet Communications Experiment", *ARRL 7th Computer Networking Conference*, pp. 186-193, Columbia, Maryland, 1 October 1988.

[5] Price, H., and McGwier, R., "Pacsat Software", *ARRL 7th Computer Networking Conference*, pp. 145-149, Columbia, Maryland, 1 October 1988.

[6] Johnson, L., and Green, C., "Microsat Project - Flight CPU hardware", *ARRL 7th Computer Networking Conference*, pp. 186-193, Columbia, Maryland, 1 October 1988.

PACSAT Data Specification Standards

Harold E. Price, NK6K
Jeff Ward, G0/K8KA

ABSTRACT

This document provides a standard way of describing PACSAT data formats in specifications, and provides certain assumptions for implementors.

Purpose

This document describes the standard format for PACSAT data.

Background

This standard is based on the following assumptions:

1) The spacecraft are the critical resources in the PACSAT/groundstation network. If a particular data representation can conserve memory space and CPU cycles in the spacecraft, all other items being equal, the representation that favors the spacecraft should take precedence.

2) The UoSAT and the AMSAT-NA PACSAT hardware are based on an Intel 80186-compatible device. Therefore, all internal multi-byte numeric data is stored with the least-significant byte in low-order memory.

3) The UoSAT and the AMSAT-NA PACSAT software is largely based on the Microsoft C programming language.

4) The UoSAT and the AMSAT-NA PACSAT software development systems are based on IBM PCs or compatibles.

Discussion

The primary decision to be made in PACSAT data formats is "big endian" (BE) vs. "little endian" (LE). Most network standards are defined as BE, meaning the Most Significant Byte (MSB) of multi-byte data appears in low order address space, and the Least Significant Byte (LSB) appear in high order memory. The UoSAT and Microsat spacecraft all use Intel 80186 or compatible CPUs, which store data with the LSB first, and are LE.

Multi-byte data appears in many places in PACSAT data, including the file headers and the control structures of the broadcast and FTL0 protocols. If these protocols were BE, the spacecraft software would need to swap byte order in several places. Whether done as in-line code or as function calls, these conversions use both CPU cycles and code space. It is clear that a native data representation will result in a more efficient utilization of the spacecraft CPU, and that the data format conversions, if any, should be done on the ground. Experimentation was done showing that avoiding byte swapping on the spacecraft resulted in significant space savings.

This will not affect the actual high-level software code, as prudent programmers who wish to write transportable code that is applicable to BE and LE hosts will use macro calls to swap the byte order when moving data from an external source to local variables. By using the somewhat less common LE in the protocol specification, the macro will be active on BE systems when it would normally be active on LE systems. In any case, the macros would still be present in the source file.

For example,

```
fnum = NETSWAP32(broadcast_head.fnum);
```

would be the line of code to read in the file number from a broadcast protocol frame. This code will be the same no matter which order the protocol required the 4-byte integer field to be in.

Taking these assumptions into account, the standard to be used when defining data exchange formats between PACSAT and a ground station are as defined below.

Intended Applicability

This document is primarily intended to apply to shared file formats, such as the standard PACSAT File Header; and to PACSAT specific protocols such as the PACSAT Broadcast Protocol. It is not meant to infer that existing protocols, such as IP, are to have integers byte-swapped when transmitted to a PACSAT.

PACSAT Data Structure Specification Standard

1) All structure definitions in PACSAT standards documents should provide C structures wherever possible to describe data formats.

2) All structures are assumed to be packed; do not assume slack bytes are provided to align words and doublewords.

3) All multi-byte numeric data is assumed to be stored and transmitted with the Least Significant Byte first.

4) Where it is necessary to number bits, the least significant bit is zero.

5) The standard method for referring to hexadecimal constants will be the C standard 0xhh.

6) The assumed length of an unsigned or int type is 16 bits.

7) The "left" end of a string is stored and transmitted first.

8) "ASCII" characters are the printable ASCII characters 0x20-0x7f.

9) Times are represented by the UNIX 4-byte unsigned integer counting the number of seconds since 0000 UTC 1 January, 1970.

Pacsat Protocol: File Transfer Level 0

Jeff Ward, G0/K8KA
Harold E. Price, NK6K

ABSTRACT

This document specifies Version 0 of the File Transfer Level 0 (FTL0) protocol designed for use on store-and-forward satellites (PACSATs). The protocol provides procedures for transferring binary files to and from a server computer using an error-corrected communication link.

Further to basic file transfer facilities, FTL0 provides file selection and directory procedures. Because a server may contain many files, only some of which are of interest to each client, the client may "select" a subset of the server's files. Subset selection is based on a flexible logical equation involving a large number of file characteristics. The scope of directory requests or file download requests can be limited to those files currently selected.

This protocol is designed to work specifically with files in which the actual "data" or "body" of the file is preceded by a standard "header", specified in the PACSAT File Header Definition document.

1.0 BACKGROUND

Terrestrial packet-radio bulletin board systems (PBBSs) in the amateur radio service generally provide a character-based, human-to-machine interface through which user's command the PBBS software. A similar interface has been adopted on the FUJI-OSCAR store-and-forward (PACSAT) satellites. Through the character-based interface, the user selects messages, requests directory listings, and views, stores and deletes messages. All messages are stored as ASCII text files, and the PBBS maintains a directory entry for each message. The directory contains message source, destination, title, time of storage, size, and status. The PBBS also stores data relating to each user station, such as the most recent message each user station has listed or viewed.

Since messages and PBBS commands contain only ASCII characters, the user's station equipment can be as basic as an ASCII terminal connected to a terminal node controller (TNC). This simplicity makes the current system attractive, and has lead to its widespread adoption.

These current PBBS systems also have several drawbacks, stemming from the text-based user interface and from the lack of a flexible, informative message header. All messages must be in ASCII, only simple ASCII message headers are supported, and only a few search-keys are provided for message selection.

These limitations are particularly damaging in the PACSAT environment, where the user base is large, the communications bandwidth is restricted, and the onboard computer software must be compact and highly reliable.

The File Transfer Level 0 protocol defined here, and the PACSAT File Header Definition presented elsewhere, combine to provide a PBBS protocol suite which overcomes the limitations of the current text-based protocols.

The proposed protocols are based on a different model of a store-and-forward messaging network. The PBBS is a "server" and the user stations are "clients". The server stores files, each of which contains a "header" of known format, followed by a "body". The header holds information about the file and about the body, including (but not limited to) the file length, creation date, destination, source, keywords, and body compression technique. The body is the data portion of the file,

which can be any digital information, e.g. binary, ASCII, FAX, digitised voice, etc.

The user station, or "client" is assumed to be under computer control. The client sends files to the server and receives files from the server over an AX.25 data link, using an automated server/client file transfer protocol. The client software interprets and displays the file headers and bodies for the human user. When the user wishes to upload a file to the server, client software must add suitable standardized header fields to the file, and it may also compress or otherwise prepare the file for uploading.

Each user is generally interested in only a small subset of the files available on a server. Using a powerful selection facility the client can select an interesting subset of the files on the server for directory listing or downloading. The selection can be based on any combination of items in the file header, thus user's software can assure that the user only "sees" files which are likely to be of interest.

The server might not keep a user data base, and the client software must store any important information concerning the user's last server session. For example, to implement a command equivalent to the PBBS command "list new", the client software must store the message number, or date and time of the of the last access to the server.

Since this protocol is also useful for terrestrial servers, we use the terms "server" and "client" throughout, instead of "PACSAT" and "groundstation". The protocol does depend on the presence of the standard PACSAT file header, this is to implement file locking or other features. A version of the protocol that can be used by terrestrial stations without the PACSAT headers is under consideration.

FTL0 includes several features that make the protocol somewhat more complex than some others in use in the amateur packet network. These features include simultaneous bidirectional file transfer, file locking by forwarding gateways, and multiple destination forwarding. As is sometimes the case, the actual implementation is smaller than the specification. The pseudo-code provided in Appendix B and C show that the protocol is actually quite straightforward. The authors also plan to make the prototype source code available on a public BBS.

1.1 FTL0

The FTL0 protocol can support full-duplex message transfer, in which a single data link (AX.25 connection) is used simultaneously to upload one file and to download another file. Client software does not necessarily have to implement this capability. The protocol provides for continuation of uploads and downloads interrupted by loss of the data link (generally caused by LOS at a satellite groundstation), and includes confirmation handshakes at all critical stages of file transfer. An FTL0 file transfer should withstand interruption at any stage.

It is assumed that the client software uses the transparent mode of an AX.25 TNC and that the AX.25 connection represents an error-free channel. Other data links which present an ordered, error corrected byte stream may be used. Although the link may be terminated at any time, it is assumed that the link will not pass corrupted or out-of-order bytes, and that processes at both ends of the link will be notified if the link is terminated. The PACSAT File Header standard provides overall checks on the integrity of files transferred to and from fully-compliant servers.

FTL0 commands and responses form simple "packets" inside the byte stream on the communications link. Specifically, in AX.25, an FTL0 packet may be transmitted in one or more AX.25 I frames; neither the server nor the client is aware of the AX.25 I frame boundaries. The packet format is described in 2.0 below.

1.2 Associated Documents

PACSAT Data Specification Standards defines the meaning of data types and structures within this specification.

The PACSAT File Header Definition defines the file header fields which must be pre-pended to files before uploading to a fully-compliant FTL0 server. (FTL0 procedures for terrestrial servers without PACSAT file headers are under development.)

1.3 Terminology

server	is the PACSAT or the PBBS.
client	is the user station.
commands	are transmitted from the client to the server
responses	are transmitted from the server to the client
uplink	is data flow from the client to the server
downlink	is data flow from the server to the client
packet	is a sequence of bytes as defined in 2.0 below
frame	is an AX.25 level 2 I frame
data link	generally an AX.25 level 2 link, but may be a link using any protocol which provides an ordered, error-checked, data transparent byte stream.

1.4 Overview of Operation

To allow simultaneous uploading and downloading, FTL0 clients and servers implement two state machines. One is related to file uploads, and the second to all other possible commands. FLT0 is not symmetric; the protocol state machines implemented by the server are necessarily different from those implemented by the clients.

Dividing the data stream into packets allows for modular software implementation and the multiplexing of commands and data on a single data link. The packet format is simple. Every packet begins with 2 bytes of control information, which can be followed by 0 to 2047 bytes of data. The packet may arrive in many AX.25 I frames. No packet synchronization or checksumming is provided, since the data link (AX.25 connection) provides a notionally error-free connection. Full file error checks are performed at the completion of file transfers.

It is assumed that neither the client software nor the server software is aware of or can control data link (AX.25) frame boundaries.

A FTL0 session starts when an data link is established between the server and the client. The server will send a LOGIN response when the connection is established.

To allow the user to specify a group of files for directory listings or downloading, the client software implement may implement menus or some other user interface.

Once the user has indicated the desired files, the client software converts the user's preferences into an equation , such as

((SOURCE = "G0K8KA*") && (KEYWORD = "*FTL0*")) && (FILE_NUMBER > 3215).

And then encodes the equation in a postfix, binary format as a SELECT command. When the server receives a SELECT command it builds a list of files for which the equation evaluates to TRUE. As a response to the SELECT command, the server informs the client how many files are in the selection list.

When the client sends a one of the directory commands, the server responds by transmitting directory information for the next 10 files on the selected list. Each subsequent directory command results in the server transmitting directory information for a further 10 files from the selection list, until the list is exhausted.

Specific files or files from the selection list may be requested by the client for downloading. The download procedure consists of two command/response cycles. On the first cycle, the client requests the file and the server responds by transmitting the file as a series of DATA packets followed by an END_OF_DATA packet. The server then waits for the client to acknowledge receipt of the file. When the server receives the acknowledgement, it transmits a final handshake and the download is finished.

To upload a file to the server, the client sends an UPLOAD command telling the server how large the file is. The server responds with a handshake which either tells the client to abort or to proceed. If instructed to proceed, the client sends the entire file as a series of DATA packets followed by an DATA_END packet. The server receives the entire file, and then either acknowledges the transfer or sends an error indication.

Because PACSAT may go over the horizon in the midst of a transaction, downloading and uploading can be continued during several sessions with the server. When a download is interrupted, the client must retain the file offset at which communications was interrupted. For interrupted uploads, the server stores the appropriate offset.

1.5 File Identification

Although servers may identify files using some kind of system file name (PACSATs use the MS-DOS standard of an 8 character body and a 3 character extension), FTL0 does not use server file names to identify files. Instead, the server assigns every file a 32-bit file number. This is to insure that two files with different contents but the same name will never be confused. Thus, every message which a user uploads to PACSAT (or another FTL0 server) will have a unique identifier.

1.6 Gateway Procedures

Some messages on PACSAT will be destined for gateway stations which will introduce the messages into other networks. It is essential that only one gateway download and relay each message. To accommodate this, FTL0 provides locked downloads. Only one gateway at a time can perform a locked download of a given file for a given destination. If a locked download is interrupted, this will be indicated in the PACSAT File Header of the file, so that other gateways can tell when a file is in the process of being delivered.

PACSAT File Headers allow files with multiple destinations, and separate locks are implemented for each destination. See Appendix A for further information.

1.7 Delivery Registration

A simple form of delivery registration is supported by FTL0. In the final handshake after receiving a file, the client can command the server to modify the contents of the PACSAT File Header to show the AX.25 address of the client and the time at which the download was completed. It is intended that this facility be used if the file is specifically addressed to the client station, not for files of general interest. If the file has multiple destinations, one "registration" is supported for each destination. See Appendix A for further information.

1.8 State Variables

In order to support full-duplex operations, the server and the client maintain 2 state variables, one relating to the uploading of files and the other relating to all other operations (selecting, directories, and downloading). These are called the uplink state and the downlink state variables.

Appendix B provides pseudo-code definitions of the state transitions for servers, Appendix C is for clients.

2.0 PACKET FORMAT

An FTL0 packet is a sequence of information bytes preceded by a two byte header. After establishment of a data link, the first byte delivered to either client or server is the first byte of a packet header. The packet header informs the client/server the type of the packet and the number of information bytes which will follow. There may be between 0 and 2047 information bytes, inclusive. After reception of the final information byte of a packet, the client/server expects the next byte to be the first byte of another packet header. Such a stream is shown below, where [<data>] indicates that there may be no data bytes.

```
<length_lsb><h1>[<info>...]<length_lsb><h1>[<info>...]
|----First FTL0 packet-----|----Second FTL0 packet---|
```

2.1 Header Format

```
struct FTL0_PKT {
   unsigned char length_lsb;
   unsigned char h1;
}
```

<length_lsb> - 8 bit unsigned integer supplying the least significant 8 bits of data_length.

<h1> - an 8-bit field.

bits 7-5 contribute 3 most significant bits to data_length.

bits 4-0 encode 32 packet types as follows:

0	DATA
1	DATA_END
2	LOGIN_RESP
3	UPLOAD_CMD
4	UL_GO_RESP
5	UL_ERROR_RESP
6	UL_ACK_RESP
7	UL_NAK_RESP
8	DOWNLOAD_CMD
9	DL_ERROR_RESP
10	DL_ABORTED_RESP
11	DL_COMPLETED_RESP
12	DL_ACK_CMD

13 DL_NAK_CMD
14 DIR_SHORT_CMD
15 DIR_LONG_CMD
16 SELECT_CMD
17 SELECT_RESP

2.2 Information Length

The <information_length> value is formed by pre-pending bits 7-5 of the <h1> byte to the <length_lsb> byte. <information_length> indicates how many more bytes will be received before the beginning of the next packet. If <information_length> is 0, there are no information bytes.

3.0 LOGON

As soon as a data link is established between the client and the server, the client is considered to be logged on. The server sends a LOGIN_RESP packet with one byte of flags and a 4-byte timestamp.

When the client receives the LOGIN_RESP packet the server is initialized and ready to begin transactions.

3.1 Initiation of Session

An FTL0 session is initiated when a data link is established between the client and the server.

3.2 Server Login Response Packet

When the data link is established the server transmits a LOGIN_RESP packet.

Packet: LOGIN_RESP
Information: 5 bytes

```
struct LOGIN_DATA{
   unsigned long login_time;
   unsigned char login_flags;
}
```

<login_time> - a 32-bit unsigned integer indicating the number of seconds since January 1, 1970.

<login_flags> - an 8-bit field.

```
bit:76543210
    xxxxSHVV
```

Bit 3, the SelectionActive bit, will be 1 if there is already an active selection list for this client. The SelectionActive bit will be 0 if there is no active selection for this client already available.

Bit 2, the HeaderPFH bit, will be 1 if the server uses and requires PACSAT File Headers on all files. If the HeaderPFH bit is 1, the flag PFHserver used in the following definition should be considered TRUE.

The HeaderPFH bit will be 0 if the server does not use PACSAT File Headers. If the HeaderPFH bit is 0, the modified procedures specified in Section 7 should be used.

Bits 1 and 0 form a 2-bit FTL0 protocol version number. The version described here is 0.

3.3 Server Login Initialization

Upon transmitting the LOGIN_RESP packet the server should initialize its state variables to UL_CMD_WAIT and DL_CMD_WAIT.

3.4 Client Login Initialization

Upon receiving the LOGIN_RESP packet, the client should initialize its state variables to UL_CMD_OK and DL_CMD_OK.

4.0 SELECT

The select command is the mechanism through which the client can search the server for desirable files. The human user can search the files on the server based on any information contained in the PACSAT File Header. To achieve this goal, the implementation and specification go through several levels of abstraction which must be followed closely.

The user's desires form an equation. The equation might simply be "the file is to G0K8KA and it was stored after my last logon". Or it might be more complex, e.g. "the file is less than 8 kbytes, and it was created after 9 July 1990, and it has "landrover" as one of its keywords". Humans would naturally express this in "infix" notation:

(FILE_SIZE < 8k) && (CREATE_DATE > 9 July 1990) && (KEYWORDS = = "*landrover*")

The server uses "postfix" notation, like an RPN calculator, and wants an equation like:

(FILE_SIZE < 8k) (CREATE_DATE > 9 july 1990) && (KEYWORDS = = "*landrover*") &&

To convert a postfix logical equation into a SELECT command, the client must encode the logical operators, the relational operators the header item names, and the values to compare against. The header item names and comparison values are encoded just as they are in the PACSAT File Header definition. The logical operators and relational operators are encoded as single bytes. The complete syntax is defined below.

4.1 Client Issuing Select Command

When the client's downlink state variable is DL_CMD_OK (e.g. the client is not involved in another select, file download, or directory download), the client may transmit a SELECT_CMD packet.

Packet: SELECT_CMD
Information: variable length SELECTION

SELECTION is defined recursively by the following structures and rules.

```
struct SELECTION {
    struct LVALUEx equation;
    unsigned char end_flag;
}
```

<end_flag> - 0x00

<equation> is recursively defined by two structures LVALUE0 and LVALUE1:

```
struct LVALUE0 {
    struct LVALUEx t1;
    struct LVALUEx t2;
    unsigned char lop;
}
```

<t1> and <t2> are further LVALUE0 or LVALUE1 structures.

<lop> is and 8-bit field representing a logical operator:

```
        bit 76543210
            00000001     is AND
            00000010     is OR
```

```
struct LVALUE1 {
    unsigned char relop;
    unsigned int item_id;
    unsigned char length;
    unsigned char constant[];
}
```

<relop> is an 8-bit field encoding a relational operator:

bit 7
0

bit 654	Relational Operator
000	equal to
001	greater than
010	less than
011	not equal
100	greater than or equal to
101	less than or equal to
110	reserved
111	reserved

bit 3210	Type of Comparison
0000	treat <constant> as a multi-byte unsigned integer (valid for <length> 1,2, and 4).
0001	treat <constant> as a multi-byte signed integer (valid for <length> 1,2,and 4).
0010	treat <constant> as an array of unsigned char
0011	treat <constant> as an array of ASCII characters, convert to lower case before making comparison.
0100	treat <constant> as an array of ASCII characters, convert to lower case before comparison and interpret wildcard characters.

ALL OTHER VALUES RESERVED.

<item_id> is a 16-bit unsigned integer identifying one of the PACSAT Header Definition header items.

<length> is the number of bytes in the <constant[ ]> array.

<constant[ ]> is an array of bytes which are compared against the header item identified by <item_id>, using the specified relational operator and comparison type.

4.2 Response to Select Command

The server responds with one of the packets from 4.2.1 or 4.2.2 and returns its downlink state to DL_CMD_OK, ready to accept another command from the client.

4.2.1 Successful Selection

If the server can interpret the SELECT_CMD packet, it responds with a SELECT_RESP packet

Packet: SELECT_RESP
Information: 2 bytes
    unsigned int no_of_files

<no_of_files> is a 16 bit unsigned integer telling how many files have been selected. It may be 0.

4.2.2 Unsuccessful Selection

If the server cannot interpret the SELECT_CMD packet, it responds with a DL_ERROR_RESP packet

Packet: DL_ERROR_RESP
Information: 1 byte
    unsigned char err_code

<err_code> is ER_POORLY_FORMED_SEL if the selection equation could not be parsed by the server because of a syntax error.

5.0 DOWNLOAD

To receive a file from the server, the client uses the download command. This command can be used to download a specific file, to continue the downloading of a specific file, or to download the next file in the selection list.

5.1 Client Initiates Download

When the client's uplink state variable is DL_CMD_OK, the client may send the DOWNLOAD_CMD packet.

Packet: DOWNLOAD_CMD
Information : 9 bytes of arguments with the following structure

```
struct {
    unsigned long file_no;
    unsigned long byte_offset;
    unsigned char lock_destination;
}
```

<file_no> is a 32-bit binary integer uniquely identifying a file on the server. Two reserved values have special meanings: If <file_no> is equal to 0xffffffff, the server will attempt to use the next file in the current selection list, moving from older files to newer files. If <file_no> is equal to 0, the server will attempt to use the next file in the current selection list, moving from newer files toward older files. If <file_no> is not one of the reserved values, the server attempts to download the file which it associates with <file_no>.

<byte_offset> is a 32-bit unsigned binary integer giving the offset from the beginning of the file at which the download should begin. If the client is continuing an aborted download, this argument should be set to the number of bytes previously received. Otherwise it should be 0.

<lock_destination> is an 8 bit unsigned binary integer. It will generally be set to 0 by non-gateway stations.

[If <lock_destination> is not 0, it indicates that the client wishes to conduct a locked download for the destination numbered by <lock_destination>. If the <lock_destination> exists and is not already locked, the client's AX.25 address is placed in the PACSAT File Header AX25_DOWNLOADER item associated with the specified DESTINATION item. Destination numbering is defined in section x.x]

5.2 Server Responses to the Download Command

When it receives a DOWNLOAD_CMD packet the server determines whether the download is possible. This may include determination of the desired file number from the current select list.

5.2.1 Downlinking File Data

If the download is possible the server transmits the file data beginning <byte_offset> bytes from the beginning of the file. If <byte_offset> is greater than or equal to the file length, no data is transmitted. The file data bytes are transmitted as information in DATA packets. These packets may be any size from 0 to 2047 bytes.

[If <lock_destination> is not-zero, the server sets the <DOWNLOAD_TIME> associated <lock_destination> before beginning to send DATA packets.]

Once the server has transmitted a DATA packet containing the last byte of the file, or if the <byte_offset> was greater than or equal to the file length, the server transmits a single DATA_END packet. After transmitting the end packet, the server expects to receive a DL_ACK_CMD or DL_NAK_CMD.

5.2.2 Failure of Download Command

If the server cannot service the download command, it responds with a DL_ERROR_RESP packet.

Packet: DL_ERROR_RESP
Information: 1 byte
unsigned char err_code;

<err_code> is an 8-bit unsigned binary integer will be one of:

ER_SELECTION_EMPTY if the selection list has no more files in it or no select command was previously received.

ER_NO_SUCH_FILE_NUMBER if the file identified by <file_no> does not exist.

ER_ALREADY_LOCKED - <lock_destination> was not 0, and the file is already locked for the specified destination.

ER_NO_SUCH_DESTINATION - <lock_destination> was not 0, and the file has fewer than <lock_destination> DESTINATION items in its PACSAT File Header.]

5.3 Client Accepting or Rejecting Downloaded File

5.3.1 Successful Download

Upon reception of the DATA_END packet from the server, the client performs any possible checks on the received file. If the file passes the checks, the client transmits a DL_ACK_CMD packet.

Packet: DL_ACK_CMD
Information: 1 byte
unsigned char <register_destination>

<register_destination> is an 8-bit unsigned integer identifying one of the destinations in the PACSAT File Header of the downloaded file. <register_destination> should be 0 if the client does not wish to register receipt of the file.

Upon receiving the DL_ACK_CMD packet, the server completes the download process.

[If <register_destination> is not 0, the appropriate DESTINATION item in the PACSAT File Header is located and the associated AX25_DOWNLOADER and DOWNLOAD_TIME items are updated. If <lock_destination> was non-zero in the DOWNLOAD_COMMAND, the server locates the appropriate DESTINATION item in the PACSAT File Header and updates the associated DOWNLOAD_TIME item.]

5.3.2 Unsuccessful or Aborted Download

If the client wishes to abort the download before receiving the DATA_END packet from the server, or finds that the downloaded file fails some integrity test after reception of the DATA_END packet, the client sends the DL_NAK_CMD packet.

Packet: DL_NAK_CMD
Information: none

Upon receiving this packet, if the server has not already sent the DATA_END packet, it will do so and proceed to the final unsuccessful download handshake (5.4.2). If DL_NAK_CMD is received after the server has transmitted the DATA_END packet, the server proceeds to the final download handshake.

5.4 Final Download Handshake

5.4.1 Completion of Successful Download

[If the server receives the DL_ACK_CMD, it removes any <lock_destination> lock from the file and sets <AX25_DOWNLOADER> and <DOWNLOAD_TIME> for the specified <DESTINATION> item.]

[If <register_destination> in the DL_ACK_CMD is non-zero, the server attempts to set the <AX25_DOWNLOADER> and <DOWNLOAD_TIME> items associated with the specified <DESTINATION> item. If the

<register_destination> does not exist, the server transmits a DL_ABORTED_RESP packet.]

If <register_destination> was 0, the server transmits a DL_COMPLETED_RESP packet, and returns its downlink state variable to DL_CMD_OK.

Packet: DL_COMPLETED_RESP
Information: none.

This response tells the client that the server has completed processing the DL_ACK_CMD. The client downlink state variable should be set to DL_CMD_OK.

If the L2 link fails before the client receives the DL_DONE_RESP, the client cannot be sure the DL_ACK_CMD was processed. This is important if the client had specified a <lock_destination> or a <register_destination>. In either of these cases, the client should treat the download as incomplete and continue it later. This assures that the PACSAT File Header has been properly modified. If both <lock_destination> and <register_destination> were 0, the client can consider the download complete.

5.4.2 Completion of Unsuccessful Download

A download is "unsuccessful" if the server receives the DL_NAK_CMD from the client or if <register_destination> is non-zero and indicates a non-existent destination.

If <lock_destination> was non-zero, the lock is removed and <DOWNLOAD_TIME> is updated.

The server transmits the DL_ABORTED_RESP packet.

Packet: DL_ABORTED_RESP
Information: none

The server downlink state variable returns to the DL_CMD_OK state.

Upon reception of the DL_ABORTED_RESP the client downlink state variable returns to DL_CMD_OK.

6.0 DIRECTORY

The client uses directory commands to get information about files on the server. Servers conforming to the PACSAT File Header Definition send PACSAT File Headers as directory entries. "Long" directory entries are the complete PACSAT File Header and "short" directory entries are only the Mandatory File Header items.

The client can request a directory entry for a specific file or for the files in the selection list. The selection list can be scanned from oldest to newest files or from newest to oldest files.

6.1 Initiating a Directory

The client may request a directory whenever the client downlink state variable is DL_CMD_OK. To request a directory, the client transmits either a DIR_LONG_CMD packet or a DIR_SHORT_CMD packet.

Packet: DIR_LONG_CMD
or DIR_SHORT_CMD
Information: 4 bytes
unsigned long file_no;

<file_no> is a 32-bit unsigned binary integer identifying a file or files on the server. There are two reserved values: 0 and 0xffffffff. If <file_no> is 0xffffffff, the server will send directory entries for the next 10 files in the current selection list, moving from older files to newer files. If <file_no> is 0, the server will send directory entries for the next 10 files in the current selection list, moving from newer files toward older files.

6.2 Server Responses to Directory Commands

6.2.1 Successful Directory Request

If the directory request can be serviced (there are files in the selection list, or a specified file exists) the server will transmit one or more PACSAT File Headers in a series of DATA packets followed by a DATA_END packet. A maximum of 10 file headers will be transmitted for each DIR command.

6.2.2. Failed Directory Request

If the directory request cannot be serviced, a DL_ERROR_RESP packet is transmitted.

Packet: DL_ERROR_RESP
Information: 1 byte
    unsigned char err_code;

<err_code> will be one of:

ER_SELECTION_EMPTY - if the <file_no> was 0 or 0xffffffff, and either there are no files left in the selection list or there is no selection list.

ER_NO_SUCH_FILE_NUMBER - if <file_no> is not 0 or 0xffffffff and no file identified by <file_no> exists on the server.

7. UPLOAD

7.1. Initiating an Upload

The client can initiate an upload any time the client's upload state variable is UL_CMD_OK.

Packet: UPLOAD_CMD
Information: 8 bytes

```
struct {
    unsigned long continue_file_no;
    unsigned long file_length;
}
```

<continue_file_no> - a 32-bit unsigned integer identifying the file to continue. Used to continue a previously-aborted upload. Must be 0 when commencing a new upload.

<file_length> - 32-bit unsigned integer indicating the number of bytes in the file.

7.2 Server Responses to Upload requests

Downlink Packet: UL_GO_RESP or UL_ERROR_RESP

7.2.1 Successful Upload Request

Packet: UL_GO_RESP
Information: 8 bytes

```
struct {
    unsigned long server_file_no;
    unsigned long byte_offset;
}
```

<server_file_no> is a 32-bit unsigned binary integer identifying the client's file on the server.

<byte_offset> is a 32-bit unsigned binary integer number of bytes from the beginning of the file at which the client should begin file transmission. This will be 0 if the <continue_file_no> was zero.

After receiving the UPLOAD_PROCEED_RESP packet, the client should advance to the data transmitting state described in Section 7.3.

7.2.2 Unsuccessful Upload Request

If the server cannot process the upload request, it will transmit an UL_ERROR_RESP packet.

Packet: UL_ERROR_RESP
Information: 1 byte
    unsigned char err_code;

<err_code> must be one of:

ER_NO_SUCH_FILE_NUMBER if <continue_file_no> is not 0 and the file identified by <continue_file_no> does not exist. Continue is not possible.

ER_BAD_CONTINUE if <continue_file_no> is not 0 and the <file_length> does not agree with the <file_length> previously associated with the file identified by <continue_file_no>. Continue is not possible.

ER_FILE_COMPLETE if <continue_file_no> is not 0 and the file identified by <continue_file_no> was completely received on a previous upload. Note - receipt of this command should be accepted by the client as confirmation of file receipt by the server.

ER_NO_ROOM if the server does not have room for the file.

After transmitting the UL_ERROR_RESP packet, the server's uplink state variable is set to UL_CMD_OK.

7.2.3 Continuation of Uploads

Any file for which the client receives a UL_GO_RESP should be continued until a UL_ACK_RESP or a non-recoverable UL_ERR_RESP or UL_NAK_RESP for that file is received. The server should be prepared to continue reception of any file for which it has transmitted a UL_GO_RESP. Some file numbers will be allocated by the server and lost by link

failure before the UL_GO_RESP reaches the client; a 32-bit file number space provides sufficient scope for some lost file numbers. To avoid saving partial files for these lost file numbers, the server should not reserve space for a message until it has received at least one DATA packet from the client.

7.3. Data Uplinking Stage

The client now uplinks the bytes from the file, in DATA packets. The uplinking should begin <byte_offset> bytes from the start of the file. After transmitting a DATA packet containing the last byte in the file, the client should transmit a DATA_END packet. The client should also transmit a DATA_END packet if the <byte_offset> was greater than or equal to the file length.

The server may attempt to terminate the upload by transmitting a UL_NAK_RESP at any time during the upload. If the client receives a UL_NAK_RESP packet before transmitting a DATA_END packet the client must send a DATA_END packet.

7.4. Completion of Upload

7.4.1 Successful Upload Completion

When the server receives the DATA_END packet it will check the integrity of the file as far as possible. If the checks pass, the server will downlink a UL_ACK_RESP packet.

Packet: UL_ACK_RESP
Information: none

After transmitting the UL_ACK_RESP the server uplink state variable is UL_CMD_OK. After receiving the UL_ACK_RESP, the client uplink state variable is UL_CMD_OK.

7.4.2 Failure Caused by Server Rejecting Upload

The server may reject an upload while the client is sending DATA packets (due to file system problems on the server) or after the client has sent the DATA_END packet (due to corruption of the file).

If the server must abort the upload while receiving DATA packets or after receiving the DATA_END checks fail, it sends the UL_NAK_RESP packet.

Information: 1 byte
unsigned char err_code;

<err_code> must be one of:

ER_BAD_HEADER - The file either has no PFH, or has a badly-formed PFH.

ER_HEADER_CHECK - The PFH checksum failed.

ER_BODY_CHECK - The PFH body checksum failed.

ER_NO_ROOM - The server ran out of room for file storage before the upload was complete. The server will implement procedures to avoid frequently running out of room, but this cannot be guaranteed.

After transmitting the UL_NAK_RESP packet, the server uplink state variable is UL_CMD_OK. After receiving the UL_ERROR_RESP, the client uplink state variable is UL_CMD_OK.

7.4.3 Link Failure During Upload

If the data link fails before the server receives the DATA_END packet, the server retains the offset of the next byte needed for the file. This value will be transmitted by the server as <byte_offset> if the client later continues the upload by specifying the <server_file_no> in the <continue_file_no> of an UPLOAD_CMD packet. If the offset is not 0, the server may safely retain a temporary file containing the data received so far. If the server retains files for which the <byte_offset> is 0, these should be purged after some reasonable time, since the client may not know the <server_file_no> of the file.

8.0 TERMINATION OF SESSION

The session is terminated by closing the data link. Either the client or the server may decide to close the link when uplink state is UL_CMD_OK and downlink state is DL_CMD_OK. The link is also terminated if unexpected packets are received.

If the link terminates when any state machine is not in the _CMD_OK state, appropriate actions are taken.

APPENDIX A - Use of the PACSAT File Header

The PACSAT File Header Definition defines a standard header which will be found at the beginning of every file downloaded from PACSAT (or any fully compliant FTL0 server).

One purpose of the PACSAT File Header is to allow clients to determine the status of files on the server. Specifically, the originating client may wish to find out if the file has been downloaded by its intended destination, and a Gateway client may wish to find out if a message still needs to be downloaded for relay to a particular destination. Both of these tasks are done by examining a set of PACSAT File Header Items comprising <DESTINATION>, <AX25_DOWNLOADER> and <DOWNLOAD_TIME>.

A gateway will examine <DESTINATION> to see if it can relay or deliver the file. If so, it will examine <DOWNLOAD_TIME> and <AX25_DOWNLOADER> to determine the delivery status of the message:

If <DOWNLOAD_TIME> is zero, the message still needs to be relayed.

If <DOWNLOAD_TIME> is non-zero, but <AX25_DOWNLOADER> is blanks, the message is in the process of being downloaded, and the download started at <DOWNLOAD_TIME>.

If <DOWNLOAD_TIME> is non-zero, and <AX25_DOWNLOADER> is non-blank, the message has been downloaded for relay to the <DESTINATION>.

A similar (but simpler) procedure is used to see if a specific destination station has "registered receipt" of a file. When receipt is registered (at the end of a successful download), <AX25_DOWNLOADER> and <DOWNLOAD_TIME> will both be correctly set.

For these checks to work correctly, the downloading clients must use the proper command procedures. Gateways must set the correct <lock_destination> in their DOWNLOAD_CMD packets, and clients must set the correct <register_destination> in the DL_ACK_CMD. The <lock_destination> procedure should be implemented by all gateways. The <register_destination> procedure is optional for client implementations.

The proper number for <register_destination> or <lock_destination> is determined as follows. If a file has only one <DESTINATION> item, it is destination 1. Any other destinations are numbered sequentially in order of occurrence in the PACSAT File Header.

APPENDIX B - Server State Transition Definitions

SERVER STATE DEFINITIONS

The following pseudo code defines the transitions of the Server state machines in FTL0. One state machine is used for file uploading. The other state machine is used for all other procedures.

It is assumed that the data link will be terminated if there have been no packets transmitted or received for a period of time (TBD). If the link is terminated by this "activity timeout" both state machines receive a "Data Link Terminated" event. "Data Link Terminated" is also generated if the link fails for any reason.

When an unexpected packet is received by one of the state machines, that state machine terminates the data link and returns to an uninitialized state. The second state machine is notified of this through a "Data Link Terminated" event, and also returns to an uninitialized state. At this point, the FTL0 processing for this client ceases until the establishment of another data link.

STARTUP

State variables are created when a data link is established between client and server. The server executes the following:

```
Transmit LOGIN_RESP packet.
ul_state := UL_CMD_OK
dl_state := DL_CMD_OK
```

SERVER UPLINK STATE MACHINE

This machine receives only frames which are relevant to the file uploading process : UPLOAD_CMD, DATA, and DATA_END. It also is executed upon timeout of the input timer or termination of the level 2 link (from local or remote causes).

```
state UL_CMD_OK {

    EVENT: Receive UPLOAD_CMD packet{
        if ok to upload
            Send UPLOAD_GO_RESP packet.
            ul_state <- UL_DATA_RX
        else
            Send UL_ERROR_RESP packet.
            ul_state <- UL_CMD_OK
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul_state <- UL_UNINIT
    }
}

state UL_DATA_RX {

    EVENT: Receive DATA packet {
        Try to store data.
        if out of storage {
            Transmit UL_NAK_RESP packet.
            Close file.
            Save file_offset and file_number.
            ul_state <- UL_ABORT
        }
        else {
            Update file_offset.
            ul_state <- UL_DATA_RX
        }
    }

    EVENT: Receive DATA_END packet {
        Close file.
        if file passes checks {
```

```
            Transmit UL_ACK_RESP packet
            ul_state <- UL_CMD_OK
        }
        else {
            file_offset <- 0
            Save file_offset and file_number
            Transmit UL_NAK_RESP packet
            ul_state <- UL_CMD_OK
        }
    }

    EVENT: DEFAULT{
        if (file_offset > 0)
            Save file_offset and file number.
        if EVENT is not "data link terminated"
            Terminate data link.
        ul_state <- UL_UNINIT
    }
}

state UL_ABORT {

    EVENT: Receive DATA packet
        ul_state <- UL_ABORT

    EVENT: Receive DATA_END packet
        ul_state <- UL_CMD_OK

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul_state <- UL_UNINIT
    }
}
```

SERVER DOWNLINK STATE MACHINE

This machine receives all packets other than UPLOAD_CMD, DATA, and DATA_END.

```
state DL_CMD_OK {

    EVENT: Receive SELECT_CMD packet {
        if syntax is correct {
            Form select list.
            Transmit SELECT_RESP packet.
            dl_state <- DL_CMD_OK
        }
        else {
            Transmit DL_ERROR_RESP packet.
            dl_state <- DL_CMD_OK
        }
    }
```

```
    EVENT: Receive DIR_LONG_CMD or DIR_SHORT_CMD packet {
        if there are directories to send
            dl_state <- DL_DIR_DATA
        else {
            Transmit DL_ERROR_RESP packet.
            dl_state <- DL_CMD_OK
        }
    }

    EVENT: Receive DOWNLOAD_CMD packet {
        if request can be serviced  {
            if <lock_destination> <> 0
                set PFH DOWNLOAD_TIME.
            dl_state <- DL_FILE_DATA
        }
        else {
            Transmit DL_ERROR_RESP packet.
            dl_state <- DL_CMD_OK
        }
    }

    EVENT: Receive DL_NAK_CMD packet
        dl_state <- DL_CMD_OK

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- DL_UNINIT
    }
}

state DL_DIR_DATA {
    if there is more directory data to send {
        Transmit directory data in DATA packets.
        dl_state <- DL_DIR_DATA
    }
    else {
        Transmit DATA_END packet.
        dl_state <- DL_CMD_OK
    }

    EVENT: Receive DL_NAK_CMD packet{
        Transmit DATA_END packet.
        dl_state <- DL_CMD_OK.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- DL_UNINIT
    }
}
```

```
state DL_FILE_DATA {

    if there is more file data to send
        Transmit file data in DATA packets.
        dl_state <- DL_FILE_DATA
    }
    else {
        Transmit DATA_END packet.
        dl_state <- DL_FILE_END
    }

    EVENT: Receive DL_NAK_CMD packet {
        If lock_destination < > 0
            unlock lock_destination.
        Transmit DATA_END packet.
        Transmit DL_ABORTED_RESP packet.
        dl_state <- DL_CMD_OK
    }

    EVENT: DEFAULT {
        If lock_destination < > 0
            unlock lock_destination.
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- DL_UNINIT
    }
}

state DL_FILE_END {

    EVENT: Receive DL_NAK_CMD packet {
        dl_state <- DL_CMD_OK
        Transmit DL_ABORTED_RESP packet.
        If lock_destination < > 0
            unlock lock_destination.
    }

    EVENT: Receive DL_ACK_CMD packet {
        if register_destination < > 0 and register_destination exists {
            set PFH AX25_DOWNLOADER
            set PFH DOWNLOAD_TIME
            Transmit DL_COMPLETED_RESP packet.
        }
        else if register_destination < > 0 and it doesn't exist {
            Transmit DL_ABORTED_RESP.
        }
        else if lock_destination < > 0 {
            unlock lock_destination.
            set PFH AX25_DOWNLOADER.
            set PFH DOWNLOAD_TIME.
            Transmit DL_COMPLETED_RESP packet.
        }
        dl_state <- DL_CMD_OK
    }
```

```
    EVENT: DEFAULT {
        If lock_destination < > 0
            unlock lock_destination.
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- DL_UNINIT
}
```

APPENDIX C - Client State Transition Definitions

Client Pseudo-code State diagrammes.

The following diagrammes indicate how an FTL0 client should react to different events when in different states.

"User Requests" come from the process using the FTL0 state machine.

Receive packets come from the data link and transmit packets go to the data link.

It is assumed that the data link will be terminated if there have been no packets transmitted or received for a period of time (TBD). If the link is terminated by this "activity timeout" the state machines receive a "Data Link Terminated" event. "Data Link Terminated" is also generated if the link fails for any reason.

UPLINK STATE MACHINE

The client uplink state machine controls the file uploading process. File downloading and all other processes are controlled by the downlink state machine.

The state of the uplink is indicated by the ul_state variable.

The uplink state machine processes the following events:

```
    EVENT: User Requests Uplink
    EVENT: Data Link Terminated
    EVENT: Receive UL_GO_RESP
    EVENT: Receive UL_ERROR_RESP
    EVENT: Receive UL_ACK_RESP
    EVENT: Receive UL_NAK_RESP
```

Where the "EVENT: DEFAULT" is indicated, this includes all events on the above list which have not already been processed.

```
State UL_UNINIT {

    EVENT: User Requests File Upload
        Refuse.

    EVENT: DEFAULT
        ignore.
}
```

```
State UL_CMD_OK {

    EVENT: User Requests File Upload {
        Transmit UL_CMD packet, setting <continue_file_no> if necessary.
        ul_state <- UL_WAIT
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul_state <- UL_UNINIT
    }
}

State UL_WAIT {

    EVENT: Receive UL_GO_RESP packet. {
        Associate <server_file_number> with the file.
        Mark the file CONTINUE.
        ul_state <- UL_DATA
    }

    EVENT: Receive UL_ERROR_RESP packet. {
        If error is unrecoverable, mark the file IMPOSSIBLE.
        ul_state <- UL_CMD_OK
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul_state <- UL_UNINIT
    }
}

State UL_DATA {
    If there is data to send {
        Transmit DATA packet.
        ul_state <- UL_DATA.
    }
    else {
        Transmit DATA_END packet.
        ul_state <- UL_END.
    }

    EVENT: Receive UL_NAK_RESP packet. {
        if error is unrecoverable mark file IMPOSSIBLE.
        Transmit DATA_END packet.
        ul_state <- UL_CMD_OK.
    }
```

```
    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul_state <- UL_UNINIT
    }
}

State UL_END {

    EVENT: Receive UL_ACK_RESP packet. {
        Mark file COMPLETED.
        ul_state <- UL_CMD_OK
    }

    EVENT: Receive UL_NAK_RESP packet. {
        If error is unrecoverable mark file IMPOSSIBLE.
        ul_state <- UL_CMD_OK
    }
}
```

DOWNLINK STATE MACHINE

The client's downlink state machine processes the following events:

```
    EVENT: User requests download
    EVENT: User requests directory
    EVENT: User requests selection
    EVENT: User requests abort
    EVENT: Receive DL_ABORTED_RESP packet.
    EVENT: Receive DL_COMPLETED_RESP packet.
    EVENT: Receive SELECT_RESP packet.
    EVENT: Receive LOGIN_RESP packet.
    EVENT: Receive DATA packet.
    EVENT: Receive DATA_END packet.
    EVENT: Receive DL_ERROR_RESP packet.
```

The only user request which generates and event when not in DL_CMD_OK state is "user requests abort", which is processed in the DL_DATA state. All other requests are ignored (perhaps queued) until dl_state is DL_CMD_OK.

```
State DL_UNINIT{
}

State DL_CMD_OK{

    EVENT: User requests download {
        Transmit DL_CMD packet.
        dl_state := DL_WAIT.
    }

    EVENT: User requests directory {
        Transmit DIR_LONG_CMD or DIR_SHORT_CMD packet.
        dl_state := DL_DIR_WAIT.
    }
```

```
    EVENT: User requests selection {
        Transmit SELECT_CMD packet.
        dl_state := DL_SEL.
    }

    EVENT: User requests abort {
        ignore.
        dl_state := DL_CMD_OK
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}

State DL_WAIT{

    EVENT: Receive DL_ERROR_RESP packet {
        dl_state <- DL_CMD_OK.
    }

    EVENT: Receive DATA packet. {
        Store data.
        Mark file INCOMPLETE.
        dl_state <- DL_DATA.
    }

    EVENT: Receive DATA_END packet. {
        If file at client is ok {
            Mark file DATA_OK.
            Save <continue_offset> equal to file length.
            Transmit DL_ACK_CMD.
        }
        Else {
            Mark file INCOMPLETE.
            Save <continue_offset> of 0.
            Transmit DL_NAK_CMD.
        }

        dl_state <- DL_END.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}
```

```
State DL_DATA{

    EVENT: User requests abort {
        Save current offset as <continue_offset>.
        Transmit DL_NAK_CMD.
        dl_state <- DL_ABORT.
    }

    EVENT: Receive DATA packet. {
        Store data.
        dl_state <- DL_DATA.
    }

    EVENT: Receive DATA_END packet. {
        If file is ok at client
            Transmit DL_ACK_CMD packet.
        Else
            Transmit DL_NAK_CMD packet.
        dl_state <- DL_END
    }

    EVENT: DEFAULT{
        Save current offset as <continue_offset>.
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}

State DL_END{

    EVENT: Receive DL_ABORTED_RESP packet. {
        dl_state <- DL_CMD_OK.
    }

    EVENT: Receive DL_COMPLETED_RESP packet. {
        Mark file COMPLETE.
        dl_state <- DL_CMD_OK.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}

State DL_ABORT{

    EVENT: Receive DATA packet. {
        Ignore.
        dl_state <- DL_ABORT.
    }
```

```
    EVENT: Receive DATA_END packet. {
        dl_state <- DL_END.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}

State DL_DIR_WAIT{

    EVENT: Receive DATA packet. {
        Send directory data to user.
        dl_state <- DL_DIR_DATA.
    }

    EVENT: Receive DL_ERROR_RESP packet. {
        dl_state <- DL_CMD_OK.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}

State DL_DIR_DATA{

    EVENT: Receive DATA packet. {
        Send directory data to user.
        dl_state <- DL_DIR_DATA
    }

    EVENT: Receive DATA_END packet.{
        dl_state <- DL_CMD_OK
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}
```

```
State DL_SEL{

    EVENT: Receive DL_ERROR_RESP packet. {
        dl_state := DL_UNINIT
    }

    EVENT: Receive SELECT_RESP packet. {
        dl_state := DL_UNINIT
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl_state <- UL_UNINIT
    }
}
```

APPENDIX D - Error Codes

Value	Name
1	ER_ILL_FORMED_CMD
2	ER_BAD_CONTINUE
3	ER_SERVER_FSYS
4	ER_NO_SUCH_FILE_NUMBER
5	ER_SELECTION_EMPTY
6	ER_MANDATORY_FIELD_MISSING
7	ER_NO_PFH
8	ER_POORLY_FORMED_SEL
9	ER_ALREADY_LOCKED
10	ER_NO_SUCH_DESTINATION
11	ER_SELECTION_EMPTY
12	ER_FILE_COMPLETE
13	ER_NO_ROOM
14	ER_BAD_HEADER
15	ER_HEADER_CHECK
16	ER_BODY_CHECK

PACSAT Broadcast Protocol

Harold E. Price, NK6K
Jeff Ward, G0/K8KA

ABSTRACT

The case for a broadcast protocol for use on PACSATs is made, and a suitable protocol is proposed. In the proposed protocol, files of general interest are chopped into <UI> frames and repeatedly sent by the PACSAT in round-robin fashion or on request. Each <UI> frame datagram contains enough information for groundstation software to place the frame in the correct position in the appropriate file. Information indicating the type of file being received is also included in each frame.

A protocol for groundstations to request retransmissions of specific frames is included.

1.0 Background

PACSAT is a generic term used to describe a digital store-and-forward satellite mission in the Amateur Radio Service. Two of four Microsats and one of two UoSATs launched in January 1990 will have PACSAT as a primary missions.

PACSATs will use the AX.25 frame as the basic link layer protocol, either in the full AX.25 connected mode, or as unconnected <UI>-frame datagrams.

A PACSAT will have several types of data to send. Some of these are:

1) Forwarded mail messages. These are messages are not destined for PACSAT as an endpoint, but are in transit between forwarding gateway stations.

2) Personal electronic mail messages. These are messages to and from individuals who are using the satellite as a BBS; entered either directly by a human-run connection, or by using a program that pre-formats messages for fast or unattended transmission. These messages use PACSAT as an endpoint. It can be argued that all mail should be forwarded mail, that no one use PACSAT as a direct BBS. There are three counter arguments:

a) For access in remote areas without a terrestrial infrastructure in place.

b) For emergency access by minimal ground stations.

c) To permit large numbers of people to have a direct hands-on experience with satellite communications.

3) Realtime Telemetry. These are current spacecraft telemetry values, such as solar array power, internal temperature, etc.

4) Stored Telemetry. This is a file of one or more telemetry values stored over time, for example, the output of the solar arrays once a second over the last orbit. This is usually called "Whole Orbit Data" or "WOD".

5) Bulletins. These are items of general interest, orbit predictions, AMSAT News, Gateway, etc.

6) Point-to-multipoint messages. These are messages which one PACSAT user wishes to send to several other PACSAT end users. This may be implemented using "mail groups" or "CC" lists.

Items 1) and 2) above are primarily point to pointoperations, either an individual reading his

own mail, or a gateway forwarding mail to be picked up by another gateway.

Item 3) is a self-contained frame that needs no other frames to convey its information.

The other items are germane to this discussion. Both are data types that would be of interest to a large number satellite users people. Any time that more than one user is interested in the same information, it will generally make more sense to broadcast it (send it once for reception by many) rather than send the information separately to each individual.

Current terrestrial packet radio use is highly inefficient. If 160 people in southern California want to read the 20k byte ARRL Gateway newsletter, they each log into a BBS and read it separately. This requires 3.2MB of data to be sent. Because of packet overhead, collisions, acks, etc., the actual number of bytes sent, and the total channel time, is much higher.

Assuming a very optimistic average throughput of 120 bytes/second, it takes 7.5 hours of channel time to allow 160 people to get a copy of the biweekly Gateway. Taking the standard loaded aloha 18% of that figure, 42 hours of channel time are required. Because there are 24 hours in a day, and there are seven packet channels in use on 2 meters alone in southern California, it is possible to accumulate 42 hours of channel time in less than a day. The inefficiency of the system is masked by the large amount of time and RF spectrum available.

Now assume that one wished to distribute Gateway on a Microsat, an environment where time and spectrum are at a premium. The Microsat is visible for about 14 minutes a pass, or about 60 minutes a day. It would then take 42 days to accumulate 42 hours of channel time, since even though the Microsat has multiple uplinks, it has only one downlink. UoSAT-3 (UO-14), with its 9600 baud downlink, would require only 5 days to move the same data.

Even if we assume a very disciplined set of users who access the satellite one at a time, we can move the efficiency from 18% to nearly 95%, enabling UO-14 to service the 160 users in 1.5 days, and Microsat in 8.5. This is assuming all of the single downlink frequency was devoted to the activity of downloading Gateway to individual users.

There are several conclusions that can be drawn from this. One is that a PACSAT is useless for dealing with a large number of individual users if they all want individual copies of the same file. It would be far better to have a gateway in the local area read a copy, then distribute it on the ground for everyone else. This is, however, not much different than what we have now, and we're looking to use the satellite to improve general conditions.

Another conclusion is that if there were a way to send a single copy of Gateway that could be seen by all 160 users at the same time, this would reduce both the load on the satellite, and the load on the terrestrial network. Since the satellite CAN see all 160 users in southern California at one time, the solution is now clear.

A broadcast protocol for items of general interest is clearly desirable. A strong case for one can be made in the terrestrial network as well, but the multiplicity of frequencies and the 24 hour availability of VHF/UHF networks makes the advantages harder to see. The single downlink of a PACSAT, and the tyranny of orbital mechanics makes the need more evident in the satellite environment. In a broadcast mode, UO-14 can transmit at least 3.2 million bytes of data to a station in the mid latitudes in an average day, AO-16 and LO-19 can each send at least 432,000. That is a lot of mail.

There is another advantage to a broadcast protocol, and again, it is more telling in the satellite service. A broadcast protocol, being one way, does not require a transmitter at the receiving site. The complexity of the ground station is reduced as the 2 meter transmitter and its antenna are not required to receive the broadcast.

Finally, the PACSAT environment gives another encouragement to broadcast protocols: because the transmit and receive frequencies are on different bands, it is inherently full-duplex. Since PACSAT will be the only transmitting station on the downlink, a major source of data-loss, collisions with other stations, is removed. When the link is good, there is no need for retransmissions, making the broadcast protocol more efficient than the normal ACK/timeout AX.25 protocol.

2.0 Attributes of a Broadcast Protocol

The AX.25 protocol, in its connected mode, makes the following guarantees:

1) All bytes are received once, in the order they were sent, with no gaps.

2) No bytes are received in error (within the limits of CRC16).

This means that the two stations can establish a connection, and then send a file. When the expected number of bytes or an end of file marker is received the receiving station can assume that it received the entire file (or message) correctly.

The general philosophy is that the sending and receiving stations work together to retransmit each frame (or small group of frames, 1-7) repeatedly until it is received correctly, before moving on to the next frame.

The connected mode requires two-way point to point transmissions, and is not suitable for a broadcast protocol.

The unconnected mode of AX.25 make these guarantees:

1) Byte order is maintained within a frame only.

2) No bytes in a frame are received in error (within the limits of CRC16).

One or more frames may be missing within a group of frames. Although some TNCs allow you to receive frames with CRC errors, you can receive no reliable indication that you have missed a frame. AX.25 does not provide frame sequence numbers in these frames. You can not know if the frame you have just received immediately follows the previously received frame, or if you missed some.

Although on average, the new PACSATs have the strongest amateur satellite signals to date, the receiving station will still get occasional dropouts from local conditions, polarity reversals due to the spacecraft's changing orientation, 70cm radar, etc. Loss of signal as the spacecraft goes over the horizon will also cause a loss of frames.

All of this means that if you receive a file as a set of broadcast <UI> frames, AX.25 itself does not provide enough information to allow you to tell if the entire file has been received, or if it has gaps. A broadcast protocol must provide this missing information. The broadcast protocol wouldride inside a standard AX.25 <UI> frame.

Ideally, the broadcast protocol would have the following attributes:

1) Any frame, when received independently, can be placed in the proper location within the file to which it belongs.

2) When the all frames have been received, the receive station can tell that the file is complete.

3) For file types where it makes sense, partial files should be usable. For example, if a data compression scheme is used, the file should be able to be incrementally decompressed, e.g., a 20000 bytes of a 20256 byte file should not be useless simply because the first 256 bytes are missing.

To carry this to full frame independence, this means that compression algorithms should be such that the decompression of one frame does not depend on the receipt of any other frame. This may require the use of a less-efficient compression algorithm for files which are to be incrementally decompressed, but partial files, and perhaps all of the information, can be extracted from a smaller set of received frames.

Files that are not meant to be incrementally decompressed may use any compression scheme, the broadcast protocol maintains data transparency.

Four rules can be derived from the above desires:

1) Each frame must contain a file identifier

2) Each frame must contain something that identifies this frame's position in the file.

3) The file must contain a special record that defines file attributes, particularly file size. Other items of interest would be the actual file name, creation date, etc.

4) If a partial file is to be usable, especially if compression is used, each frame must contain enough information to allow use of a partial file.

2.0.1 Additional Error Protection

It is most likely that TNCs operating in KISS mode will be used to receive the UI frames which make up PACSAT Broadcasts. The authors' experiments indicate that the link between the KISS TNC and the user's personal computer may be prone to errors caused by a lack of flow control. This is especially true when using high radio data rates such as UoSAT-OSCAR-14's 9600 baud link.

Although the AX.25 CRC assures that only correctly received frames will be passed on by the KISS TNC to the host computer, we are now not certain that the frame reaching the host computer is error free. If we process incorrectly received frames, even occasionally, files will be lost or corrupted. Thus, we must add some error protection to the broadcast frame.

After experimentation, a 16-bit CRC was selected.

2.1 Frame Header Contents

The above rules define a frame structure that looks like this:

<flags><file_id><file_type><offset><data><crc>

Each field is discussed below.

2.1.1 file_id

At first glance, the simplest file id would be the actual name of the file. Since the PACSAT file system permits 8 bytes of name and 3 bytes of extension, this would lead to 11 bytes of overhead per frame, or 4%. This is somewhat large. There is also the problem that the contents of a file named "error.log" may change, and so the file name is not truly a unique file identifier.

To overcome this problem PACSAT assigns each created file a "file number", which is 4-bytes long and will uniquely identify the file. This number is used as the file_id in the Broadcast Protocol header.

The receiving station will not know the actual name of the file until the file header record is received. This record could be transmitted more frequently than other records, increasing the chances that it would be available in any partially-received file.

2.1.2 offset

Each frame of broadcast data must contain the position in the file where the data came from. Each frame, then, contains an offset field. The offset is the byte number of the first byte in that frame, relative to the first byte in the file. For PACSAT files, byte 0 is always the first byte of the PACSAT File Header.

To place a received data frame into a file, one need only:

```
lseek((long) offset);
write(handle, data, frame_size);
```

While it is easy to insert the frame bytes into the file, it is much harder to know when all the bytes have been received. There are several methods, one is maintaining a list of holes in the file, much as some TCP/IP implementations do when reassembling IP packets.

The size of the offset field is important, too small and the size of broadcast files are limited, too large and the overhead for each file is increased. We've chosen 24 bits as a compromise between efficiency and maximum file size. This allows for files up to 16Mb.

2.1.3 Flags

The flag field is a byte which provides option bits. Two bits are currently defined. "Length" says that an option length field is present in the frame header. "EOF" says that this is the last frame of the associated file.

2.1.4 File_type

If a partial file, which could be just a single frame, is to be useful, some information on the file type must be provided with each frame. Examples of file types are:

ASCII text bulletins
Images from UoSAT-E CCD

Stored telemetry
Digitized voice
Machine-ready Keplerian element updates
Incrementally decompressable files
Non-decompressable files

The file_type can also inform the groundstation software that a file is incrementally decompressable, or not compressed at all.

The file_type byte is the same byte which is found in the file header record in the file. File types are assigned and defined in a separate document.

2.1.5 data

This part of the frame is the file data, compressed or uncompressed. The file_type can be used to tell what type of data is present.

2.1.6 CRC

Reception errors in the AX.25 UI frame can be detected using the 16-bit CRC which is part of the HDLC frame, and this low-level error detection would, ideally, be enough to protect the Broadcast protocol frames. However, in reality, the UI frames are usually received by a KISS TNC, and the link between the KISS tnc and the host computer may be unreliable (especially at high speed) due to the lack of flow control in the KISS standard. For this reason, we have chosen to append a 16-bit CRC to the broadcast frame WITHIN the AX.25 UI frame data field. This CRC will be calculated using the XMODEM CRC specification, which has been successfully and efficiently implemented on many microcomputers.

2.1.7 File Header

Clearly, the 8-bit file_type and the 32-bit file_id cannot convey all of the information which a groundstation needs to know about a file (e.g. time of creation, complete file name, file size). This information is in a structure at the beginning of each file. The file header may be of variable length, but would contain an offset to the first user data byte. The header is always at start of the file, with a byte offset of 0.

A proposal for a standard PACSAT File Header is provided in a separate document.

2.2 Binary Data

For greatest efficiency, all header information is coded in binary. This is a departure from previous amateur digital satellite philosophy.

Currently, UO-9, AO-10, UO-11, and AO-13 transmit telemetry and bulletins in uncompressed Ascii. This was, in part, to make this information available to the widest possible audience. The audience was a late 1970/early 1980 audience, and was assumed to have only the modem and a terminal, with little or no computer assistance. Telemetry on all of the above satellites can be read in Ascii and decoded with pencil and paper. The amount of telemetry generated is small, and the broadcast data is bi-weekly bulletins.

PACSATs are being designed for a 1990s environment. A new piece of hardware is required, either an external TNC, or an add-on card for your computer. In almost all cases, a computer is present at the ground station. The number of users who must deal directly with the raw data is reduced, perhaps to zero.

An increased emphasis on compressed data will lead to downlink which is mostly binary anyway. Several ground station programs are planned or already available which will allow users to monitor the downlink and acquire the data. The authors plan to make public domain source and shareware programs available for the IBM PC, with proceeds going to AMSAT-NA and AMSAT-UK.

3.0 PACSAT Broadcast - How it works in practice

PACSAT command stations would upload files tagged with a broadcast rotation priority and an expiration date. On-board programs would also be able to tag or build files with this data. An on-board broadcast administrator task maintains a list of active broadcast files and assigns a file_id. Files would be broadcast based on their rotation priority, i.e., files with low priority would be sent less often that those with high priority. Files under some threshold priority would only be sent during otherwise idle downlink time, those above the threshold would be mixed in with whatever else is on the downlink. Since parsing binary data (the header) in the standard monitor mode of TNCs is non-trivial, we can assume that the KISS mode, or

other host mode will be used to implement the ground program. Therefore the PID byte in the frame can easily be used to identify broadcast protocol frames. The exact value is 0xbb.

Ground users would run a program that monitors the downlink for frames. This program would probably use the KISS mode of the TNC. As each frame was received, the file_id and offset field of the frames are used to build up an image of the file. Duplicate frames are discarded. Several files can be active at once.

A utility program is provided to convert the received files into a usable form. It can decompress compressed files, and handle partial files if possible. It will display the contents of the PACSAT File Header record in received files. It can also generate a hole list or bit map, as required, to send to PACSAT and request retransmission of missing parts of the file.

4.0 Extensions

So far this discussion has assumed that all files on PACSAT can be divided into two groups: broadcast files which are sent with the broadcast protocol, and point-to-point files which are sent using AX.25 connected mode. There is also a gray area: point-to-multipoint files, which are not of interest to all PACSAT groundstations, but might be of interest to more than one station in the same PACSAT footprint. For instance, a message to all TCP/IP users would not warrant a continuous broadcast, but it might be downloaded by several groundstations. One would like to arrange things so that when the first groundstation downloads "TCPIP.UPD", other users in the broadcast-listen mode would also receive and avoid downloading it themselves (if they're interested). There are two ways of achieving this: by using the broadcast mode when the first groundstation requests the file, or by embedding broadcast mode datagrams in the connected-mode frames.

Using the first method, the user would connect to PACSAT and request that a file or files be put in the broadcast rotation. The user would then disconnect and use his broadcast-listen program to receive the desired files. (The user could even avoid connecting in the first place by transmitting the file request in a <UI> frame on the uplink.)

Files added to the broadcast list this way would be sent at a high priority for the length of an average pass.

Users who do not receive the file completely while it is being broadcast can send a description of the missed data (hole-list) to the PACSAT and those blocks would placed in the broadcast rotation. Retransmission requests for a particular file can come from many stations, but by the nature of the round-robin broadcast, requests are not likely to be synchronized and cause a mass uplink collision.

Thus, we use the broadcast mode for all but explicitly point-to-point files.

The alternative is for the PACSAT to connect to one station and commence standard AX.25 transfer, but embed enough information into the <I> frames to allow "eavesdropping" stations to build up copies of the file being transferred. Thus, to every <I> frame PACSAT adds the standard broadcast mode header, for the benefit of stations which may be listening in on the connection. Unfortunately, the connected mode user has no idea where each <I> frame begins and ends, so his software cannot strip out the unwanted header information. Thus, we set the L bit in the <flags> field and place frame length in the <length> field. Now the connected mode user knows where each frame begins and can strip out the header.

This may stick in the gullet of pure layered protocol designers. In defense of the proposal we offer the facts that (1) the broadcast <UI> frame listener already relies on knowing where the level-2 frames begin and end; and (2) we can look on our frame format as a higher-level packet format, and it just so happens that each packet fits in one frame (for the benefit of the "eavesdroppers").

5.0 Incremental Decompression

Incremental decompression is the ability to decompress partial files. In the most general case, any frame can be decompressed independently of any other frame.

It should be noted that the desire for incremental decompression has several deleterious effects. Primarily, it forces the use of non-optimal compression techniques.

Some compression schemes, such as Huffman coding, require that a table be sent along with the file that provides deciding information. The file is not decodable without this table, so a partial file that did not include the table would be useless. To allow use of Huffman-style coding, files would have to be compressed with a small number of standard tables that can be encoded in the file type.

Many compression methods, including Huffman coding, turn a file into tokens of variable bit length. That is, a token will not necessarily be an integral number of octets long, whereas an AX.25 frame must be an integral number of octets long. An arbitrary frame can be decoded only if the start of the frame is also the start of a token, and we know the number of meaningful bits in the frame.

Also, the program doing the broadcast framing may not be the program that did the compression. This would require, for instance, that PACSAT decompress the file as it broadcasts it, so it could properly align frames and tokens.

Decompression of partial files is an area for further discussion. To allow maximum flexibility in future implementation, the proposed standard includes the possibility of variable length frames. If the length bit in the flag byte is set, the two bytes following the standard frame header are the number of valid bits in the block. This allows for bit tokens of a non-integer number of octets.

It should be noted that the WO-18 image transmission format is a form of broadcast with incremental decompression, allowing even a single frame of the compressed image to be placed in it proper location on the display screen.

6.0 Broadcast Advantages

Receive-only stations with just an omni antenna and an eavesdrop program can accumulate bulletins, telemetry, and other items of general interest just by listening.

For transmit and receive stations, if one other station is interested in the same file, then overhead is reduced by at least 100%. This should justify the additional few percent overhead of the broadcast information.

7.0 Broadcast Disadvantages

Nearly everything broadcast from PACSAT would have several bytes of binary data in the front of each frame. This would make it difficult to monitor with just a TNC and terminal. Since the assumption is made that the data will also be compressed, a computer and proper program would be required in any case, so it is unlikely there will be any TNC-only stations. We also note that most of current fleet of spacecraft are already transmitting telemetry in raw binary form.

Due to the difficulty of parsing binary data from the standard monitor mode of TNCs, only TNCs with the KISS protocol, or other host-mode protocol, will be suitable. It may be possible, depending of the availability of true 8-bit transparency, to use the monitor mode on some TNCs, since a known, fixed string would precede each monitored frame, e.g., ">QST-1:".

8.0 Implementation Status

UO-14 has been broadcasting files using the protocol defined in Appendix A. since mid summer. AO-16 and LO-19 should begin using this format by the end of August.

9.0 Proposed Broadcast Protocol

The protocol specification is provided as Appendix A to this paper.

10.0 Correspondence

Address comments to:

Telemail:	HPRICE or UOSAT
Compuserve:	71635,1174
Packet:	NK6K @ WB6YMH
	or G0K8KA @ GB2UP
Internet:	71635.1174
	@COMPUSERVE.COM
Mail:	Jeff Ward
	UoSAT Unit
	University of Surrey
	Guildford, Surrey GU2 5XH
	UK

Appendix A.

A.1 PACSAT Broadcast Protocol Description

This protocol describes a method where files can be sent from a PACSAT to an unknown number of groundstations, using the unconnected <UI> frame mode of AX.25. The protocol could also be used in purely terrestrial applications.

Each file that is to be broadcast is assigned a 32-bit file id. This number must be unique among all other files broadcast by this station. A station must be able to later match the file id with the file if the station is to handle retransmission requests.

When a file is broadcast, it is broken down into frames. Each frame is identified with enough information so that the data can be placed in the proper location and in the proper file by a receive-only groundstation. Each frame is sent within an AX.25 <UI> frame, and is totally contained within that <UI> frame.

Some files are automatically retransmitted enough times over several passes so that the likelihood of a station receiving all parts of the file at least once with no errors is high. A facility is also provided to allow a ground station to request the transmission of certain frames to allow fills of missing data.

The PACSAT Broadcast Protocol has two elements, the File Transmission Frame, use to send data; and the File Request Frame, used to request retransmission of all or part of a file.

A.2 File Transmission frame format

A.2.1 Frame

A broadcast frame consists of a header, data and a CRC . The header is variable length, depending on bits in the flag byte. The data may also be of variable length.

The broadcast frame is wholly contained within a standard AX.25 <UI> frame. The total length of the broadcast frame may not exceed the length of a legal AX.25 <UI> frame.

A <UI> frame containing a broadcast frame uses a PID of 0xbb.

A <UI> frame containing a broadcast frame uses a source address of the transmitting station, and a destination address of QST-1.

A.2.1.1 Frame Header

Each frame contains a frame header, which occupies the first n bytes of the frame. The structure of the frame header is as follows:

<flags><file_id><file_type><offset>[<length>]

This structure follows the PACSAT Data Specification Standards as regards field size and byte order, which is defined in a separate document.

```
struct FRAME_HEADER {
    unsigned char flags;
    unsigned long file_id;
    unsigned char file_type;
    unsigned int offset;
```

```
    unsigned char offset_msb;
    };
```

or

```
struct FRAME_HEADER_EXT {
    unsigned char flags;
    unsigned long file_id;
    unsigned char file_type;
    unsigned int offset;
    unsigned char offset_msb;
    unsigned int length;          /* only present if L bit set */
    };
```

flags A bit field as follows:

7	6	5	4	3	2	1
*	*	0	V	V	0	L

L 1 length field is present
0 length field not present

E 1 Last byte of frame is the last byte of the file.
0 Not last.

VV Two bit version identifier. This version is 0.

0 Always 0.

* Reserved, must be 0.

file_id A number which identifies an active broadcast file. All frames which are part of the same file are tagged with this number.

file_type The file_type byte from the file header. Provided so that partial files received without the header can be decoded. File types are defined in a separate document.

length If the L bit is set, this field is present it in the header. It is the number of bits that are to be used in the data field. This field has two intended uses: when variable length blocks are used with the broadcast carried inside a higher level protocol (so that the frame length is lost), and when a non-integer number of octets of data are used.

offset If the O bit is not set, this is the block number of the block. If the O bit is set, this is the offset from the start of the file for the first byte in this frame. This field is the lower 16 bits of a 24 bit integer.

offset_msb The high order 8 bits of the offset.

A.2.1.2 Data

If the block mode is used, the length of the data must be fixed. It may be any length, but the following must be true:

file_offset_of_first_byte_in_frame = offset * this_frame_size

All frames in a particular transmission of a file should be the same length, to avoid having the receiver resize his bit map too often.

If byte numbers rather than block numbers are used, the data may be any length. If the length field is present, the data must contain only that number of bits, rounded up to the next octet boundary.

A.2.1.3 CRC

A two-byte CRC as defined by the XMODEM protocol follows the data. The crc covers all data bytes in the AX.25 UI frame, including the broadcast frame header.

An inefficient definition routine for checking the XMODEM CRC when receiving a frame is:

```
unsigned short crc;             /* global crc register.              */
crc = 0;                        /* clear crc register                */
for (all received data bytes)
    gen_crc(rx_byte);           /* crc the incoming data bytes       */
gen_crc(1st_crc_byte);          /* crc the incoming first crc byte   */
if(gen_crc(2nd_crc_byte))       /* crc the incoming second crc byte  */
 bad_crc();                     /* non-zero crc reg. is an error     */
else
 good_crc();                    /* zero crc is a good packet.        */
/* Function to take a byte and run it into the XMODEM crc generator.  */
/* Uses a global CRC register called ''crc''.                          */
gen_crc(data)
char data;
{
    int y;
    crc ^= data << 8;
    for(y=0; y < 8; y++)
        if( crc & 0x8000)
            crc = crc << 1 ^ 0x1021;
        else
            crc <<= 1;
    return crc;
}
```

Further information on generation of XMODEM CRC via more efficient byte-wise methods can be found in BYTE Magazine, September 1986, pp. 115-124.

A.3 File Retransmission Request Format

Each request to retransmit portions of a file is sent in a retransmission request frame. If more data is being requested than will fit in one request, multiple requests may be sent.

This protocol is intended to service retransmissions of already-broadcast files. The manner is which a file is first requested to be broadcast is a more complex topic, including, presumably, the ability to request files based on name, date, content, keywords, mail message copy list, etc.

A.3.1 Request Frame

A request frame consists of a header and data. The header is fixed length. The data depends on the type field in the header, and can be of variable length.

The request frame is wholly contained within a standard AX.25 <UI> frame. The total length of the broadcast frame may not exceed the length of a legal AX.25 <UI> frame.

A <UI> frame containing a request frame uses the same PID as a broadcast frame, 0xbb. The source address is the address of the transmitting station, the destination address is the address of the station to which the request is directed. Thus, if a frame has the PACSAT Broadcast Protocol PID , if the destination is QST-1, it is a broadcast frame, otherwise it is a request frame.

A.3.2 Request Format

A.3.2.1 Header

The format of request is:

<type><file_id><data>

```
struct REQUEST_HEADER {
     char flags;
     unsigned long file_id;
     int block_size;
     };
```

flags - A bit field as follows:

7	6	5	4	3	2	1
*	*	1	V	V	C	C

CC Two bit field as follows:
- 00 start sending file <file_id>
- 01 stop sending file <file_id>
- 10 Frame contains a hole list.

VV Two bit version identifier. This version is 0.

1 Always 1.

* Reserved, must be 0.

block_size Requests that the broadcast use this value as a maximum size.

file_id File id of the requested file.

A.3.2.1 Data

If the CC field is 2, a hole list is present.

The format of the hole list is pairs of start addresses and lengths.

```
struct PAIR {
     unsigned int offset;
     unsigned char offset_msb;
     unsigned int length;
     };
```

The length of the hole list is determined by received frame size.

A.4 PROCEDURES

A.4.1 Identification of a broadcast protocol data frame.

The broadcast protocol will use a special PID 0xbb. Broadcast frames with be sent with a source callsign of the station's call, and a destination callsign of <QST-1>.

A.4.2 File transmission

Multiple broadcast files may be transmitted simultaneously, frames from one file can be interleaved with another. Frames can be transmitted in any order and repeated at any time.

A.4.3 File completion

The file header contains the total number of bytes in the file. Once the ground station has received that number of bytes, the file is complete. The file header also contains checksums which can be used to verify the correct reception of the file.

Appendix B.

B. Legalities Within the Amateur Satellite Service

The use of the words "broadcast" and "compression" sometimes raise the hackles of certain members of the amateur community. "Broadcast" is mentioned in the Amateur Rules (Part 97 of the FCC regulations), and compression is sometimes equated with encryption.

B.1 Broadcast

To establish the bona-fides of a broadcast protocol:

97.113(d)(2) specifically permits information bulletins consisting solely of subject matter relating to amateur radio.

The prohibited items are communications intended to be received directly by the public (1200/4800/9600 baud PSK HDLC should take care of that), or for newsgathering for broadcast purposes.

B.2 Compression

97.117 specifically permits the use of abbreviations or signals where the intent is not to obscure the meaning but only to facilitate communications. Compression using published algorithms to increase data throughput would thus be permitted.

Pacsat File Header Definition

Jeff Ward, G0/K8KA
Harold E. Price, NK6K

ABSTRACT

A flexible encoding method for PACSAT file headers is described, and "Mandatory", "Extended" and "Optional" Headers are defined. These headers are supplied by the programs which send files and/or messages to PACSAT, and by on-board programs which build files/messages intended for downloading. PACSAT file headers are present in all files stored on PACSAT.

1.0 BACKGROUND

PACSAT is a file and message switch, a BBS and a data generating device. Files may be generated by onboard processes such as telemetry gathering programs, SEU monitor programs, or imaging cameras. Files will also be used to hold the messages in the PACSAT on-board BBS. Files and messages will be sent and received by many nodes: forwarding gateways, individual user stations, command stations, and various on-board tasks. To conserve on-board storage space and communications link time, files may be compressed by a variety of compression methods.

To ensure that these files can be properly identified and processed, each file stored on PACSAT will begin with several header items. Some header items will be present on every PACSAT file; these are described below as the Mandatory Header. Another group of items must be present on all files which contain "messages"; these are described as the Extended Header. Additional special-purpose or user-defined items are described under the Optional Header.

The primary objectives of the PACSAT File Header standard are to

(1) encode all header items in a standardized manner;

(2) maintain complete separation between the file/message header and the file/message body;

(3) provide for expansion through easy incorporation of additional header items.

1.1 Overview of the PACSAT File Header System

Every PACSAT file will start with the byte 0xaa followed by the byte 0x55. This flag is followed by the rest of the PACSAT File Header (PFH). A valid PFH contains all of the items of the Mandatory Header (Section 3), and it may also contain all items of the Extended Header (Section 4) and any number of Optional Header items (Section 5). All HEADER ITEMS are encoded using a standard syntax, described in Section 2.

The PFH is terminated by a special header item, after which the file body begins.

Thus, there are 3 forms of PACSAT file header:

<0xaa><0x55><Mandatory hdr><Hdr end>

<0xaa><0x55><Mandatory hdr><Extended hdr><Hdr end>

<0xaa><0x55><Mandatory hdr><Extended hdr>[<Optional Items> . . .]<Hdr end>

2.0 PACSAT HEADER ITEM SYNTAX

All PACSAT file header items follow a single format, simplifying both specification and implementation of the PACSAT File Header. The format is:

<id><length>[<data> . . .]

2.1 <id>

The id is a 2-byte integer in which the bits have the following meaning:

bit 15	0 this is an system-defined item. 1 this is an experimental, user defined item.
bits 0-14	form the 15-bit unsigned binary number identifying the item.

The <id>, allows some 32,000 system-defined and 32,000 user defined items.

<id> *like all multi-byte integers* is stored least-significant byte first. Refer to the PACSAT Data Specification Standards document for further information.

2.2 <length>

This field is an 8-bit unsigned binary integer giving the number of <data> bytes present. Even if the size of the data item is fixed, the length is still present.

2.3 <data>

The <data> bytes may hold any type of information.

Encoding rules for system-defined items are found in this document. User-defined items may adopt any internal encoding agreed by all users of the item.

2.4 Presentation

The PACSAT File Header must always be transmitted without data compression, even if compression is applied to the body of the attached file.

2.5 Header Termination

The end of the PACSAT File Header will always be indicated by a header item with <id> 0 and <length> 0. The byte sequence is 0x00 0x00 0x00.

3.0 THE PACSAT MANDATORY HEADER

The first two bytes of a PACSAT file should always contain 0xaa followed by 0x55 to confirm that the file contains a PACSAT file header.

The 0xaa, 0x55 sequence must be followed immediately by all items of the Mandatory Header.

Mandatory Header items must be present in order of ascending value of <id>.

When preparing files for uploading to PACSAT, groundstations must initialize header items as specified below.

3.1.1 file_number

id : 0x01
length : 4
data : unsigned long file_number

<file_number> is a 4-byte unsigned serial number assigned to a file by PACSAT when the file is created. This number uniquely identifies any file.

Since the PACSAT file system makes no distinction between files and "messages", the file number is analogous to a message serial number.

INITIALIZATION - Must be initialized to 0.

3.1.2 file_name

id : 0x02
length : 8
data : char file_name[8]

<file_name> is the base name of the file as it is stored in the PACSAT file system. If the name is shorter than 8 characters, it is extended on the right with ASCII spaces (0x20).

INITIALIZATION - Must be initialized to 8 ASCII spaces (0x20), allowing PACSAT to choose its own name for the file. The <user_filename>

Optional item can be used to communicate the file's native name to another user.

3.1.3 file_ext

id : 0x03
length : 3
data : char file_ext[3]

<file_ext> is a 3 character file name extension. If the extension is shorter than 3 characters, it is extended on the right with ASCII spaces (0x20).

INITIALIZATION - Must be initialized to 3 ASCII spaces (0x20), allowing PACSAT to choose its own name for the file. The <user_filename> optional item can be used to communicate the file's native name to another user.

3.1.4 file_size

id : 0x04
length : 4
data : unsigned long file_size

<file_size> is a 4-byte unsigned integer indicating the total number of bytes in the file, including the HEADER_FLAG, all HEADER_FIELD structures, and the file body.

INITIALIZATION - Correct <file_size> must be provided.

3.1.5 create_time

id : 0x05
length : 4
data : unsigned long create_time

<create_time> is a 4-byte unsigned integer time-stamp telling when the file was created. This time-stamp counts the seconds since Jan 1, 1970.

INITIALIZATION - If <create_time> is initialized to 0, PACSAT will set the time upon receiving the file header. Otherwise PACSAT does not alter this item.

3.1.6 last_modified_time

id : 0x06
length : 4
data : unsigned long last_modified_time

INITIALIZATION - If <last_modified_time> is initialized to 0, PACSAT will set the time upon receiving the file header. Otherwise PACSAT does not alter this item.

3.1.7 seu_flag

id : 0x07
length : 1
data : unsigned char seu_flag

Files stored on PACSAT may experience Single-Event Upsets, caused by radiation. <seu_flag> is an unsigned 8-bit integer for which 3 values are currently defined:

0 means there have been no Single Event Upsets detected in this file.

1 means that one or more correctable Single Event Upsets have occurred and been corrected in this file. It will be possible, though unlikely, that multiple SEU-caused bit errors in a file block will cause an improper correction. An overall file checksum is provided as additional protection.

2 means that an uncorrectable corruption was detected in this file.

INITIALIZATION - this item must be initialized to 0.

3.1.8 file_type

id : 0x08
length : 1
data : unsigned char file_type

<file_type> is an unsigned 8-bit integer indicating what type of data is presented in the file body. Values for this item are defined in a separate document. The value 0xff is reserved as an escape indicator, in which case an Optional item of type <file_description> must be provided.

INITIALIZATION - this item must be appropriately initialized.

NOTE - It is intended that this item be used to limit the scope of message searches, therefore, values will be defined for important types of files such as: RLI/MBL compatible single messages, RLI/MBL compatible import files, VITA-only messages, etc. See Appendix A for details.

3.1.9 body_checksum

id : 0x09
length : 2
data : unsigned int body_checksum

A 16 bit checksum formed by adding all bytes in the file body into a 16 bit variable, ignoring overflow. The <body_checksum> does *not* include the bytes comprising the PACSAT file header.

The <body_checksum> is primarily intended to detect mis-corrected multi-bit errors caused by Single Event Upsets in the PACSAT memory.

INITIALIZATION - The correct <body_checksum> must be supplied.

3.1.10 header_checksum

id : 0x0a
length : 2
data : unsigned int header_checksum

A 16 bit checksum formed by adding ALL bytes in PACSAT File Header, including the leading 0x55 0xaa sequence, into a 16 bit variable, ignoring overflow. This number is then stored as the <header_checksum>. When calculating the sum the 2 <header_checksum> data bytes are assumed to be 0, and the <body_checksum> must have already been calculated.

The <header_checksum> is primarily intended to confirm correct header reception during file transfers.

INITIALIZATION - the <header_checksum> must be correctly initialized.

3.1.11 <body_offset>

id : 0x0b
length : 2
data : unsigned int body_offset

<body_offset> provides the byte offset of the first byte of the file body. <body_offset> is taken with respect to the first byte of the file, which has offset 0. The byte at offset 0 contains the 0xaa marking the beginning of the PFH. Note also that <body_offset> is equal to the length of the PFH, in bytes.

INITIALIZATION - <body_offset> must be correctly initialized.

3.2 Mandatory Header Summary

The PFH Mandatory header will be present on every PACSAT file. When preparing to upload a file or message to PACSAT, groundstation software must create a valid Mandatory header and insert it at the beginning of the file/message.

4.0 THE PACSAT EXTENDED HEADER

The PACSAT Mandatory Header defined above is designed for file transfer. It contains sufficient information to reliably upload and download PACSAT files, including transfers spread over several satellite passes. It does not contain all the header fields which are desirable for forwarding BBS messages or for implementing a complex PACSAT end-user message system. The additional header fields needed for these tasks are placed in the PACSAT file after the Mandatory Header.

A standard set of message-related header fields called the PACSAT Extended Header is described in this section. All message files uploaded to PACSAT should contain both the Mandatory and Extended headers.

If a Extended Header is present, it must immediately follow the final item in the Mandatory Header.

If any Extended Header item is present, all must be present.

Extended Header items must be present in order of ascending value of <id>, with the exception that multiple destinations are represented by multiple occurrences of items 0x14, 0x15, and 0x16.

4.1 Extended Header Summary

The Extended Header provides necessary information concerning the source and destination of a message file. Source data is encoded in a variable-length <source> item, which can contain any type of identifier. The AX.25 address of the station which uploaded the message is also provided, along with the time at which the upload was completed. Destination data is provided in the same

format, and provisions are made to allow a single message file to have several specified destinations.

Three other items useful for PACSAT message handling are defined: compression_technique, expire_time, and priority.

4.2.1 source

id : 0x10
length : variable
data : char source[]

<source> is an ASCII string used to identify the originator of the file/message. <source> can be a mixed-case string, containing any character from 0x20 to 0x7e.

INITIALIZATION - This item should contain the address of and possibly the route to the file originator. Exact details of the use for this item must be agreed among parties using PACSAT for message forwarding.

Stations using PACSAT as their "home BBS" are requested to use the form <call> @ OSCAR<num>, e.g. G0K8KA @ OSCAR14.

VITA will devise its own addressing scheme, which should be used by VITA groundstation software.

4.2.2 ax25_uploader

id : 0x11
length : 6
data : char ax25_uploader[]

Contains the ax.25 address of the station which uploaded the file. The SSID is not included in this address. If the callsign is less than 6 characters long, it will be filled to 6 characters by appending spaces (0x20) on the right.

INITIALIZATION - No initialization required.

4.2.4 upload_time

id : 0x12
length : 4
data : unsigned long upload_time

This field tells the time at which the upload was completed. If the upload is still in progress, upload_time will be 0x0000. <upload_time> is an unsigned integer counting the number of seconds since 0000 UTC Jan 1, 1970.

INITIALIZATION - Must be set to 0.

4.2.5 download_count

id : 0x13
length : 1
data : unsigned char download_count

<download_count> is an 8-bit unsigned integer incremented each time the associated file is successfully downloaded.

INITIALIZATION - set to 0.

4.2.6 destination

id : 0x14
length : variable
data : char destination[]

<destination> is an ASCII string used to identify the originator of the file/message. <destination> can be a mixed-case string, containing any character from 0x20 to 0x7f.

INITIALIZATION - PACSAT makes no attempt to interpret this item. It must be initialized to an address which will be recognized by the station intended to download the message. When addressing messages to stations using PACSAT as a "home bbs", please use <callsign> @ OSCAR<num>, e.g. NK6K @ OSCAR16.

4.2.7 ax25_downloader

id : 0x15
length : 6
data : char ax25_downloader[6]

<ax25_downloader> is the ASCII address of the groundstation which has downloaded this file for the recipient specified in the immediately preceding <destination> item.

A <destination> item must be immediately followed by an <ax25_downloader> item.

INITIALIZATION - Must be initialized to six ASCII blanks - 0x20.

4.2.8 download_time

id : 0x16
length : 4
data : unsigned long download_time

<download_time> is the time at which the message was completely downloaded by the immediately preceding <ax25_downloader> groundstation. <download_time> is an unsigned integer counting the number of seconds since 0000 UTC January 1, 1970.

An <ax25_downloader> item must be immediately followed by a <download_time> item.

INITIALIZATION - Set to 0.

NOTE - A message may have several intended destinations. For each destination, the PFH Extended header must contain header items 0x14, 0x15 and 0x16. Multiple destinations are numbered in the order of occurrence; the first <destination> <ax25_downloader> <download_time> set is destination 0, the next destination 1, etc.

4.2.11 expire_time

id : 0x17
length : 4
data : unsigned long expire_time

<expire_time> is the time after which this file should be considered inactive. As with other timestamps, this field is an unsigned long integer counting seconds since Jan 1, 1970. Expired files may be purged by the PACSAT when more free file space is needed.

INITIALIZATION - Groundstations may set this field in uploaded files, or may leave it set to 0. If a groundstation-selected <expire_time> is within system limits, it will be retained, otherwise the PACSAT will choose its own <expire_time>.

4.2.12 priority

id : 0x18
length : 1
data : unsigned char priority

This field carries the message priority field. Higher numbers are considered higher priority than lower numbers.

INITIALIZATION - The groundstation must initialize this field. Groundstation software should exercise some control over the user's abuse of this field, so that it retains some meaning in operation! Over use of high priorities reduces the utility of this field.

5.0 OPTIONAL HEADER ITEMS

The Mandatory Header and Extended Header may be followed by any number of Optional Header items. It is intended that any expansion of the PFH definition will involve only addition of Optional Items

Optional Header items need not be presented in increasing order of <id>.

5.1 System-defined Optional Header Fields

5.1.1 compression_type

id : 0x19
length : 1
data : unsigned char compression_technique

The body of a PACSAT message may be compressed to reduce the communications bandwidth and on-board storage required for the message. Groundstations, and not PACSAT, must compress and de-compress PACSAT files.

The <compression_type> item is a 1-byte unsigned binary integer. Values are available for assignment to common compression schemes. <compression_type> 0xff is reserved as an escape code indicating that additional information is to be found in a <compression_description> item.

Currently assigned values can be found in Appendix B.

INITIALIZATION - If present, must be correctly set by the uploading station.

5.1.1 bbs_message_type

id : 0x20
length : 1
data : char bbs_message_type

This field carries the single ASCII character used to indicate message type on RLI/MBL BBS messages.

5.1.2 bulletin_id_number

id : 0x21
length : variable
data : char bid[]

The <bid> item holds an ASCII string uniquely identifying the file/message. This field is used by terrestrial BBSs to stop the duplication of flood bulletins.

INITIALIZATION - PACSAT will not itself initialize <bid> on an uploaded file. It is the responsibility of the uploading station to initialize this field, if the message is a bulletin intended for introduction into the Amateur Radio PBBS network.

5.1.3 title

id : 0x22
length : variable
data : char title[]

This field carries the ASCII string message title. Most messages will have a <title>, initialized by the user to indicate the contents of the message. In some systems, this is called the Subject.

5.1.4 keywords

id : 0x23
length : variable
data : char keywords[]

This field carries one or more ASCII keywords describing the file/message. Multiple keywords must be separated by at least one ASCII space character (0x20).

5.1.5 file_description

id : 0x24
length : variable
data : char file_description[]

The <file_description> item is used only if none of the system standard <file_type> values can adequately describe the file body.

A <file_description> item is up to 255 ASCII characters describing the format of the file body. This field must be included if the <file_type> field in the Mandatory Header is set to 0xff.

For example, an uploading station might set the <file_type> to 0xff and <file_description> to "This body contains all of the files associated with a Ventura Publisher document".

5.1.6 compression_description

id : 0x25
length : variable
data : char compression_description[]

A compression_description item is used when a non-standard method of file-body compression has been used.

The item is up to 255 ASCII characters describing the method or (preferably) providing a reference to further information concerning the method. The field must be present when compression_technique in the fixed portion of the Extended File Header is set to 0xff.

For example, an uploading station might set compression_technique 0xff and compression_description to "Compressed using bmpack version 1.4, see file with title = "BMPACK specification".

5.1.7 user_file_name

id : 0x26
length : variable
data : char user_file_name[]

This field is used by groundstations using PACSAT as a file switch to transfer named files. The originating station places the desired file name in a user_file_name field, and the destination station uses this field as the name of the file after it has been received.

This field is included in addition to the file_name field because the file_name field is strictly constrained by PACSAT (e.g. no two files may have the same file_name, and the name must be no longer than 8 characters with a 3 character extension). The file_name is used by PACSAT for internal purposes, and this item, user_file_name is used for end-to-end transparent communication of a file name.

6.0 Implementation Status

Files with these headers are currently in use on the PACSATs. Additional system header items may be added from time to time, as well as file and compression types. To suggest new standard items, contact the authors.

Address comments to:

Telemail:	HPRICE or UOSAT
Compuserve:	71635,1174
Packet:	NK6K @ WB6YMH or G0K8KA @ GB2UP
Internet:	71635.1174 @COMPUSERVE.COM
Mail:	Jeff Ward UoSAT Unit University of Surrey Guildford, Surrey GU2 5XH UK

APPENDIX A - PACSAT FILE TYPES

Unless explicitly stated, a file of any <file_type> can have a compressed body. If the body is compressed, its PFH must contain the optional <compression_type> item. The PFH is never compressed.

0x00	ASCII text file intended for display/printing. Not Compressed.
0x01	RLI/MBL message body. Single message.
0x02	RLI/MBL import/export file. Multiple message.
0x03	UoSAT Whole Orbit Data
0x04	Microsat Whole Orbit Data
0x05	UoSAT CPE Data
0x06	MS/PC-DOS .exe file
0x07	MS/PC-DOS .com file
0x08	Keplerian elements NASA 2-line format
0x09	Keplerian elements "AMSAT" format
0x0a	Simple ASCII text file, but compressed.
0xff	ESCAPE - indicates that the message header includes a variable- length body_description item (see below) describing the body type, or providing a reference for further information. This code will be used for new techniques, until they can be assigned a formal identifier.

APPENDIX B - PACSAT COMPRESSION TYPES (PROPOSED)

0x00	body not compressed
0x01	body compressed using PKARC
0x02	body compressed using PKZIP

There is no intent to limit compression types to the IBM-PC. The prototype implementation of the ground station software is PC based.

A Fast Switching, Wide Bandwidth Transceiver for 70-cm Operation, The DVR 4-2

by Jerry Schmitt WXØS, Phil Anderson WØXI, and Bruce Kerns

August 2, 1990, Lawrence, Kansas

After designing, building, testing, beta testing, and finally shipping DVR 2-2 transceivers in early 1990, it became apparent quickly that there was a desire by many amateurs for an even faster transceiver for the 70-cm band. Features similar to those provided for by the DVR 2-2 were requested: fast TR switching, access to the modulator and discriminator without radio modification, data output independent of squelch, receiver derived carrier detection, simple design, and easy to work on. In addition, several suggested adding a filter for wide bandwidth operation, allowing for speeds higher than 9600 baud for networking (backbone) applications.

With these inputs in mind, the following broad specifications emerged for a simple straightforward 70-cm transceiver for 1200, 9600 and 19,200 baud operation:

- two bandwidth modes of operation, narrow and wide
- two receiver modes of operation, terrestrial and satellite (AFC)
- two data ports,
  one DVR 2-2 plug compatible,
  one for 19,200 operation,
  both without audio processing (no processing)
- TTL compatible signal levels for the 19,200 port, i.e. TTL ready
- fast TR switching
- a receiver derived carrier detect
- simple
- easy to work on
- easy to modify
- fun!

After considering a number of alternatives, regulatory requirements, and available ICs, we settled on a design represented in block diagram form by Figures 1 and 2. The receiver again, like the DVR 2-2, is built around the Motorola 3362, but additional 'glue' was added. Second, in order to gain fast TR switching, the local oscillators for transmit and receive are crystal based, derived from a wide bandwidth phase locked loop consisting of a VCO, a divide by 64 counter, digital phase detector, wide loop filter and the crystal oscillators.

Block Diagrams

The receiver and transmitter portions of the unit, less local oscillator generation are denoted in Figure 1. LO generation, TNC interfacing ports, TR control and modulation/demodulation are represented in Figure 2. Let's start by considering the traditional transceiver section, Figure 1.

The receiver is triple-conversion and based on the Motorola 3362 'receiver on a chip,' with additions. The receiver chain, from reception to detector consists of (from left to right) receive PIN TR switch, RF preamplifier and first mixer, and the 3362 receiver on a chip with narrow and wide bandwidth filters for the third IF, 455 KHz. As with the DVR 2-2, the second IF is at 10.7 MHz, thereby using conventional, easily sourced parts. The transmitter is straightforward (from top right to left), consisting of VCO at frequency, an amplifier and PIN switch to provide for LO to both transmitter and receiver, and two stages of power amplification, followed by the transmit PIN switch at the antenna port.

Push-to-talk (PTT) via the mic input or from either data port controls the position of both the antenna and LO PIN switches. A front panel switch denoted by narrow/wide (N/W) determines which 455 KHz IF filter is

Figure 1 – DVR 4-2 Transmitter and Receiver

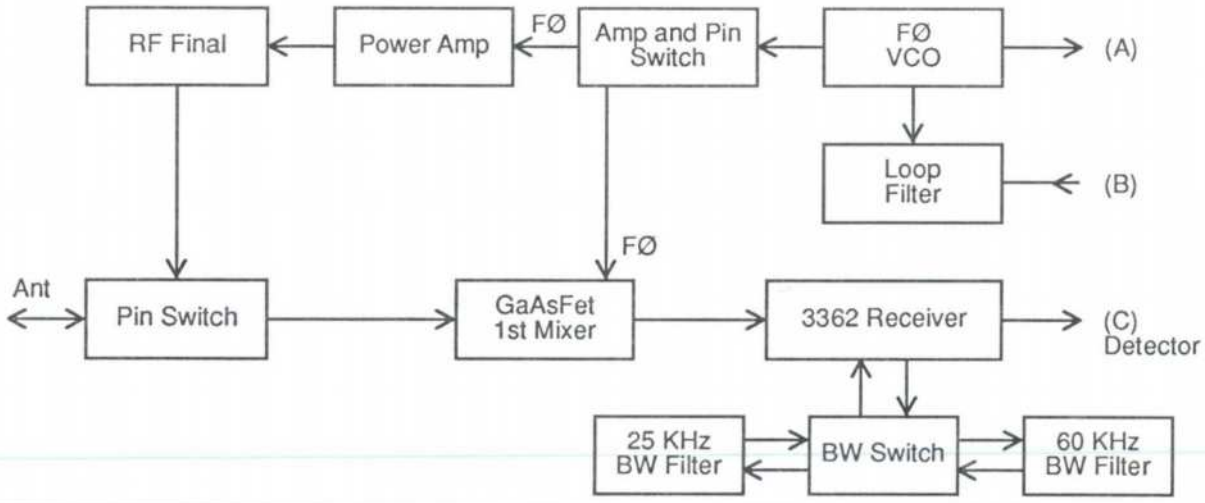
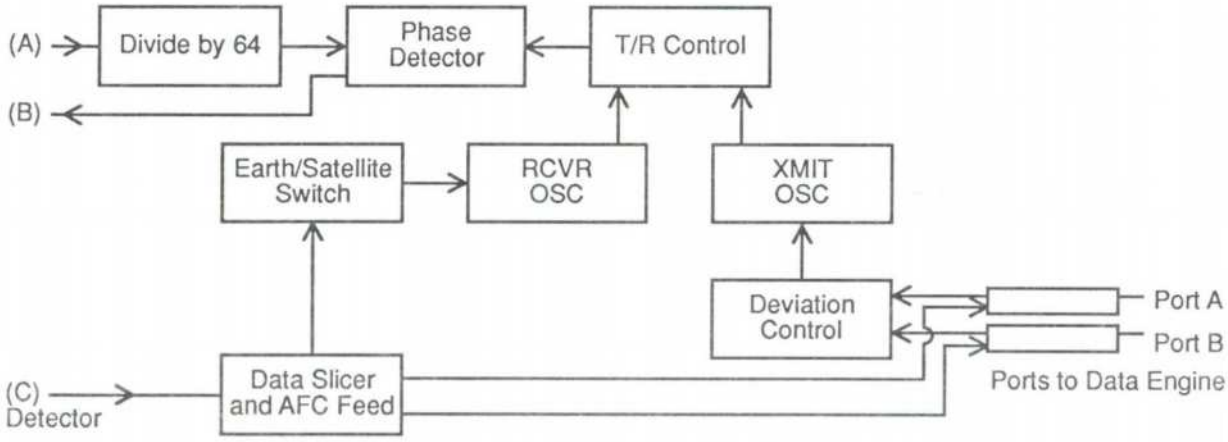


Figure 2 – DVR 4-2 Oscillators, AFC and Data Modulators



operational. In the narrow position, a Murata CFW455C is being used in the prototype at this point, providing a 6dB bandwidth of +/-12 KHz. In the wide position, a discrete stagger-tuned 60 KHz BW linear phase filter is configured. An analog switch is utilized to switch between these filters.

Now consider Figure 2 which includes frequency loop and TR control, AFC control, and modulation/demodulation data port circuits. The oscillator for the first mixer and the transmitter is derived from a loop consisting of receive and transmit crystal oscillators, digital phase detector, a divide by 64 counter, loop filter, the VCO (shown in Figure 1), and offset adjust potentiometers. The VCO frequency will be 45 MHz below the selected UHF operational frequency during reception, set to 64 times the receive crystal oscillator frequency. When PTT is asserted, the VCO frequency will shift to 64 times the transmit crystal oscillator frequency. Loop characteristics are such that TR switching is kept below 5 ms. One potentiometer per crystal oscillator was added and set to provide a control voltage for the VCO, via an analog switch, such that the loop error voltage is small for a given crystal frequency selected, providing quick lock when TR switching.

For satellite use, automatic frequency control (AFC) was added, allowing for automatic tracking of signals received with a doppler

frequency shift. The AFC receive mode is activated by setting the 'terrestrial/satellite' (T/S) front panel switch to the S position. The control voltage for AFC is derived from the discriminator output and the LO will track the average frequency of the received signal.

The modulation and demodulation functions of the DVR 4-2 are denoted at the bottom of Figure 2. From the left, the output of the receive detector feeds the data slicer. AFC control is also derived here as described above. Raw discriminator (detector) output is feed to data port A, pin for pin compatible with the DVR 2-2 DB-9 connector port. For 19,200 baud wide bandwidth operation, the output of the data slicer is converted to a TTL level and then fed to port B for interfacing to the Data Engine (TNC). A data receive clock is not generated and data scramblers for transmit and receive are not provided; these functions are assigned to the modems or partial modems that would plug inside the Data Engine or other high-speed capable TNC.

Three forms of modulation are provided for within the block labeled deviation control. As with the DVR 2-2, one may use a standard mic or the 'DVR 2-2' data interface with port A. A 50 mv p-p TX audio signal applied at port A will result in an FM signal with +/-3 KHz deviation. Port B is considered the high-speed port. A TTL TX signal applied there will generate an FM signal with +/-9600 KHz deviation. It is assumed that the 19,200 data source will be NRZ.

Intended Applications

It is anticipated, as per the requests for a wide bandwidth transceiver, that such units would be used in UHF backbone networks, probably in conjunction with packet switches. Figure 3 describes one possible implementation in conjunction with the high-speed capable Data Engine.

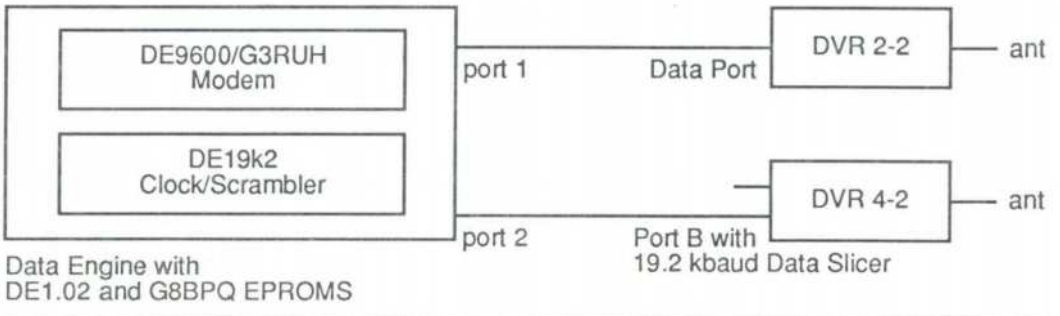
The Data Engine is shown at the left and would include a G8BPQ packet switch EPROM, a DE9600/G3RUH modem and a DE19200 clock/scrambler. A DVR 2-2 would be attached to port 1 of the Data Engine to provide 9600 2-meter operation. The DVR 4-2 would be configured for wide bandwidth operation and attached to port 2 of the Data Engine. The DE19200 clock/scrambler board would plug inside the Data Engine and provides for transmit clock, receive clock recovery and optional scrambler operation.

With the G8BPQ EPROM installed and set in multi-drop KISS mode, two more ports, each at say 1200 baud, could be added via a KPC-4 attached to the serial port of the Data Engine. The whole configuration would then provide two 1200 user access ports, one 9600 baud 2-meter backbone node and one 19200 70-cm backbone node.

Acknowledgements

The DVR 4-2 project would not have been possible without the teamwork of Jerry Schmitt WXØS, Phil Anderson, WØXI, and Bruce Kerns, and suggestions from Karl Medcalf WK5M. Additional thanks go to the beta testers and users of the first transceiver in the DVR family, the DVR 2-2. Without lessons learned from the DVR 2-2, the DVR 4-2 project may not have started.

Figure 3 – DVR 4-2 19.2 kbaud system, 440MHz



Long Distance Packet Mail via Satellite

by
Mark Sproul, KB2ICI
&
Keith Sproul, WU2Z

INTRODUCTION

The packet radio mail system is a great Electronic Mail system for amateurs. Within a local VHF area, it works fine. However, when trying to send messages farther away, they must be passed via the HF Long Haul system, which has many drawbacks and problems. Why not use satellites to bypass the HF bottleneck for a nationwide long distance mail forwarding system?

BACKGROUND

Soon after I started using packet radio, I was trying to send some messages from my home in New Jersey to my father in the south central United States. Shortly after I started sending test messages to my father, I was told not to "...waste our precious HF bandwidth with repetitive test messages..." and also that it was not possible to send messages larger than 5K through the long-haul message system because of the problems with HF packet.

The goal was to be able to send large (20K) files to him on a routine basis and ultimately be able to send binary files such as word-processor documents, and spread-sheet files, but the HF gateways would not allow this. Now I want to do something about these limitations which, in my opinion, are extremely severe.

I am not saying that the current system is bad. Actually, I like the system when it works, but that is the extent of it. The long haul HF link is the main bottle neck of the entire system. For example, I can get messages from New Hampshire to New Jersey in about 8 hours. But I get messages from Virginia that can take as long as 18 days to get to me. The strange thing about this is Virginia is closer to me than New Hampshire. Another problem is that most of the HF gateways have a file size limitation of 5K, and there are even some that have a file size limitation set at less than 2K! Since the long haul HF packet system is the main bottle neck, let's try some other way around it.

We all agree that some sort of reliable long distance packet communications is needed. Long distance packet communications has been a problem among packeteers for some time. The

HF packet system* has problems that are being worked on. This does not mean we should sit back and wait for them to be fixed. We should be experimenting with other possible means of transmitting packets over long distance.

With the orbiting amateur satellites that we have now and with the geostationary ones that are coming in the future, we can accomplish this goal. Existing satellites, such as OSCAR-13 can handle packet signals. Let us use these capabilities to supplement and eventually replace the HF Long Haul packet system.

IMPLEMENTATION

To understand this proposal, we first must understand how the Long Haul system works. When you send a message from your local BBS to someone, the BBS software checks to see where the person your message is addressed to is located. If it is someone on the local BBS, that is the end of it. If the person is at a BBS that is local to your BBS, it will get forwarded directly to that BBS. If the BBS is somewhat distant, but still accessible via local VHF packet, then the message will be transferred via one or more hops through local BBSs. If the destination is outside the local area, or 'DOMAIN', then the message gets forwarded to the local HF gateway[1]. This HF gateway is what takes care of getting your messages forwarded long distances and this is were the bottle neck is.

The way that most of the BBSs are starting to use to determine where the person is a system called "DOMAIN ADDRESSING". This is the addressing scheme that lists the BBS call, followed by the state etc, i.e. KB2ICI @ K2DLJ.NJ.USA.NA. In most cases, the lowest domain is at the state level, but in some of the larger states like California, there is another level, i.e. K6TS @ K6ABC.NCA.CA.USA.NA. In this case, the NCA is Northern California. This scheme lets us know what DOMAIN to send it to. With this, we can set up a packet forwarding system that uses satellites and NO HF stations.

Packet communications is practical with current satellites; there are even satellites that are dedicated to packet. However, this proposal does not require special packet satellites; it only requires a normal uplink/downlink satellite such as Oscar-13.

It is possible with existing equipment and software to set up a station that will automatically track, acquire signal, account for dopler shift, and make a contact, all unattended. If there were a similar station in some other state, presumably farther away than would reachable by normal VHF packet, we could then transfer packet messages from New York to California. When finished with the transfer to California, the system could then switch to another frequency where it knows there is a station in Florida. It could then transfer packet messages to Florida and

* The HF packet network is running on an STA (Special Temporary Authority) from the FCC because it has been illegal to operate an unattended HF station. This is not the case with VHF and therefore with a satellite system running on VHF/UHF frequencies.

continue for as many different states (domains) as it had messages for. The tracking software has evolved to the point were it also knows what other areas are also in the footprint of the satellite, thus able to know which states (domains) to try to contact at what time in the orbit of the satellite.

If too many of these stations were attempting this all at the same time, we would run into too many collision problems. A system would have to be set up as follows:

Each domain would have only two active stations, one for transmitting messages out of that domain, and one for receiving messages. The station for receiving messages would be the *Domain Receiving Station* and the station for transmitting would be the *Domain Transmitting Station*. These stations do not need to communicate with each other and do not even need to be close to each other. To avoid RF interference it would probably be advisable that they NOT be at the same location.

Domain Receiving Station

Each state (domain) would have ONE receiving station on a fixed frequency. Ideally each domain would have its own frequency to reside on, but this would not be required. A few frequencies with all of the domains spread among them should be quite adequate. Each domain would have an assigned frequency and adjacent domains would not be on the same frequency. At a later date more frequencies could be added and not all domains would have to be on the same satellite. Also, a station could be set up to track a primary and a secondary satellite. When the first one is not available, it could be tracking the secondary satellite.

This station would be solely responsible for receiving traffic (packet messages) into that domain. It would track its satellite as long as it was in view listening on its assigned frequency. If anything was heard, it would respond. Stations for other domains also within the footprint of the satellite could be on the same frequency. The messages would be transferred to the receiving station and then passed on to other systems via normal VHF packet channels for distribution to the final destination. This system would NOT transfer any messages out of the domain.

Domain Transmitting Station

Each domain would have ONE Domain Transmitting Station. This station would collect messages for other domains and save them. When a satellite came into view, it would try to contact another station that was also currently in the satellite's footprint. Once contact was made, all of the messages for that domain would be sent. Then it would try for the next domain that is also under the satellite's footprint. The other stations under the footprint would be constantly changing as the satellite moves. Therefore, it should be possible to contact several different domains on a single pass of a satellite.

The transmitting station would have

to have full control of the radio to be able to switch frequencies and maybe even satellite transmission modes i.e. mode L (1269/435 MHz) and mode S (435/2400 MHz). Mode B (435/145 MHz) will work fine for experimentation, but may not have enough band width for full implementation.

By implementing this concept, and using normal satellite transponders, such as OSCAR-13, New Jersey could be sending mail to Texas, Texas could be sending mail to Florida, New York to Ohio, etc., all at the same time, on the same satellite, on different frequencies.

FUTURE EXPANSION

As the implementation grew, it might become necessary to allocate one frequency for each domain, but that should not be too difficult especially on modes L and S. The speed of communications could be increased dramatically for larger amounts of messages without everyone having to get a new TNC. Only the domain stations would need the faster TNCs for more efficient passage of data.

The ideal set-up for this whole system is with geostationary satellites. If the system were implemented as described, geostationary satellites would be a natural transition. The domain transmitting station would still have to have movable antennas to be able to point at different geostationary satellites and to be able to talk to those stations that are still on the orbiting satellites. A receiving station, which only uses one satellite and one frequency would be able to be constructed with non-moving antennas pointed at the geostationary satellite it was assigned to.

OTHER PROPOSALS

Of the six MicroSats launched this past January, three are configured for store and forward packet. These are UoSAT-OSCAR-14, PACSAT-OSCAR-16, and LUSAT-OSCAR-19. Gateway stations using these store and forward orbiting BBSs have been proposed.[2] Using store and forward packet satellites is fine, however it has many drawbacks. First and foremost, it requires specialized satellites. We all know that satellites are not cheap. Also, these specialized satellites can only be used for packet, and can only be used on one frequency at a time. This limits the number of packet QSOs going on at any given instant. Also, they have to take a message in and at a later time send that message back out to someone else, thus reducing by half the theoretical through-put. The satellite BBS systems have a finite amount of storage and can easily fill up. While they will be closely monitored to prevent overflow, large amounts of traffic could make these systems unacceptably slow or cumbersome.

These satellites in low earth orbit do have their benefits, mainly a lot of people can access it with only a small investment in additional equipment. However, this proposal is focusing on the long distance back bone and as stated above, should only have two stations in each domain.

HARDWARE REQUIREMENTS FOR A STATION

The major components of a station would be as follows:

Satellite communications

- Computer controllable satellite radio
  i.e. Kenwood TS-790A,
  Yeasu 726, 736, Icom 970
- Satellite TNC as required for satellite packet
- Computer controllable rotor
- Computer for tracking satellite and frequency control of radio (this could be the same computer used for the packet operation)

Domain VHF Packet communications (local packet operation)

- VHF Packet radio
- VHF Packet TNC
- Computer for message handling (This computer must also talk to the satellite radio and the satellite control computer to coordinate the communications.)

CONCLUSIONS

A long haul packet forwarding system can be set up on satellites, totally bypassing the HF system which has some severe limitations. By limiting the number of stations per domain and a little bit of thought into frequency allocation we can set up a system that will allow any domain to send mail to any other domain within North America. Access to Europe and Asia would also be possible from some locations. This system will be able to be implemented slowly and the implementation of only a few stations will make a significant impact. It will have a very smooth transition into the future as we get geostationary satellites in place.

REFERENCES

[1] "Packet Radio's Long-Haul Mail System" Ed Juge, W5TOO *CQ The Radio Amateur's Journal,* Volume 46, Number 2, February 1990. Page 13

[2] "PACSAT Expectations and Preparations" John Branegan, GM4IHJ *The* AMSAT *Journal*, Volume 13 Number 1, March 1990

Station Traffic System: A Traffic Handler's Utility Package With Integrated Packet Support

Frank Warren, Jr. KB4CYC
Radio Amateur Telecommunications Society

The Station Traffic System (STS) is a utility program for National Traffic System (NTS) traffic handlers written in C for use in MS-DOS® and UNIX® environments providing a set of menu-selectable functions for message operations and operating environment support.

1. INTRODUCTION

STS provides a unified system for origination of formal NTS messages, generation of administrative reports, filing and storage of received formal messages, and tools to aid in the tracking and administration of the above.

2. HISTORY

STS had its origins in a BASIC program to format Net Control reports for the South Tidewater Amateur Radio Emergency Services (STARES) net on the author's Tandy Color Computer in about 1984. The Color Computer version grew to include many of the features of the current release reaching its peak in 1988.

As the author migrated to MS-DOS based systems for personal use, the decision was made to port the code to that environment. Porting was made easier because both interpreters are products of Microsoft from roughly the same timeframe.

After porting, an "If it ain't broke don't fix it!" view was taken until late 1989. BASIC's limitations with respect to serial ports and speed, as well as the author's experience with the sysop side of PBBS operations, and preference for the C language led to a rewrite of STS in C with release 3.0 (the initial C release) delivered 1 January 1990 for MS-DOS. Release 3.1 was a "Bells and Whistles" enhancement. Release 3.2 added UNIX support to the source and Message IDentification strings (MID's) on all traffic generated as packet-ready. Release 3.3 changed the rules for MID generation including switching between levels of MID generation (default ZIP Code®/postal code routed messages only) along with adding support for an extended originator field in the preamble. The current release (3.4) adds additional support for hierarchical routing for packet-ready messages and enhances message input via a set of tilde ('~') escape commands in the rewritten input routine.

3. DIVERSE ENVIRONMENTS SUPPORTED

Given the view that the amateur traffic and packet networks are parts of a larger integrated system, it is reasonable to support operation in as many operating environments as one can within reason. To this end, STS is written to operate in multiple environments and has been compiled and run in the following: MS-DOS/TURBO C®(1.5), MS-DOS/TURBO C®++(1.0), UNIX System V (SVR2 and SVR3), SunOS™(4.0.3c), Berkeley UNIX(4.3BSD), and ULTRIX™(V3.1).

4. FUNCTIONAL DESCRIPTION

STS is a menu driven program using single keystroke entry where practical. On startup if the required local data files are missing the program will prompt for the data and prepare the needed files for future use. After the initial banner the Main Menu is presented listing functions with the invoking keys. These are:

4.1 Station Activity Report [A]

This will prompt for the data required for the monthly Station Activity / Public Service Honor Roll report for transmission to the Section Traffic Manager (STM). If the STM's name, call, and home PBBS (if any) are not on file (in net.mgr) the data is prompted for and added. The report is generated as packet-ready traffic to call@PBBS if the STM's home PBBS is known.

4.2 Concatenate/Book Text Files [C]

As an aid to record keeping, this choice provides for combining existing files with optional leading comments into a new or existing file with an option of removing each of the source files once combined.

4.3 Directory [D]

Provides a short form listing of the current or specified directory.

4.4 Received Message Intake [I]

Formats and provides tracking data for messages received from sources other than one's own origination. Will prepare for filing to packet if needed data is available indicating the current station has handled the traffic while attributing the message to the proper originator.

4.5 License (Terms & Information) [L]

Redisplay of the opening screen information.

4.6 Message Origination [M]

Generate and format traffic originating at the user's station including timestamping, interrogation for special handling directives (HX instructions), and encapsulation for packet transport (if given the needed data).

4.7 Net Report (QNS) [N]

Formats a sorted post-event Net Control Station (NCS) report for transmission to a Net Manager. The report is packet encapsulated if the Net Manager's home PBBS has been included as part of the entry in the file net.mgr.

4.8 Print Files [P]

Prints files as per the underlying operating system.

4.9 Reset High Message Number [R]

Allows for resetting of the last message number used.

4.10 Display on Screen [S]

Display files to the screen like the 'more' command of Berkeley UNIX although only allowing Next, Quit, Restart, and page forward (default).

4.11 Toggle MID Generation Status [T]

Allows resetting of the conditions for attaching MID's to generated traffic as All, None, or Zip routed only. An invalid response aborts the change.

4.12 Validate Area or Postal Code vs. State [V]

This procedure allows advance checking of ZIP/Postal codes or North American Numbering Plan area codes against State/Province. This uses the same routines used to validate the data during address entry for both message origination and received message intake.

4.13 Shell Escape [!]

Spawns a sub-shell of the user's specified shell/command processor using the value of SHELL if set or else COMSPEC for MS-DOS or the Bourne Shell (/bin/sh) for UNIX.

4.14 Exit [X]

Cease operations and return to the operating system.

5. COMMAND LINE OPTIONS

STS supports setting of the status of MID generation on the command line by the inclusion of an argument of All, None, or Zip. No switch character is required and any option may be entered as only the first letter.

6. ENVIRONMENTAL VARIABLES

STS makes use of several environmental variables if available. As often as could be done use is made of variables that are also used by other programs. These are detailed below.

6.1 COMSPEC

The MS-DOS system related variable detailing the location of a copy of COMMAND.COM,

its default command interpreter.

6.2 ED

The pathname of the editor for the system to invoke when required. It should include any options or switches required to render the resulting file as ASCII text without program specific mark-up data.

6.3 HOME

A given user's initial directory in multi-user environments. HOME is the first choice for where to find user specific supporting data files. (If not set STSDIR will be used.)

6.4 SHELL

The path of the user's chosen command interpreter that will be spawned by the shell escape [!] command. Defaults: MS-DOS - COMSPEC, UNIX - /bin/sh.

6.5 STSDIR

STS specific, the directory (if not the current directory) in which to find the common supporting data files. Also the location of user specific support files if HOME is not set.

6.6 SWITCHAR

For MS-DOS systems operating with an alternative switch character (example '-') this should be set so it may be used by the program.

6.7 TFKDIR

STS specific, the directory (if not the current directory) in which to place the generated message files.

6.8 TZ

The initials for the timezone to which the system clock is set, the number of hours it lags behind UTC/GMT, and optionally the initials for daylight/summer time. Examples are: EST5EDT, CST6, MST7, PST8PDT, EDT4, or UTC0 (the default). UNIX users should not set this as it has already been set as part of system initialization. MS-DOS systems running system clocks on local time should set TZ so that the UTC based timestamps will be correct.

7. DESIGN PHILOSOPHY

In the design of the program it has been attempted to maintain a consistency of interface. The inclusion of interfaces to the graphical user interfaces supported by the various environments, in particular, have been omitted for now to give a common presentation on all directly supported operating systems. In general the program is case insensitive for single keystroke responses. The exceptions are overriding validation failures and deleting files, both of which are of a serious enough nature that accidental invocation should be avoided. For these actions the program mandates an uppercase response. As mentioned above the design of the program endeavors to make use of environmental values that can be shared by multiple programs as opposed to creating a duplicative set of such values. Additionally portions of the user interface have been designed to match the presentation of other programs the user may have experience with, most notably the use of line by line entry of text with the availability of tilde commands to allow for greater control of the process. Also data is requested by labeled prompts so that even the infrequent user will maintain awareness of the current data entry status and requirements.

8. SPECIAL FEATURES

STS supports a pair of features that have the potential to spark controversy among PBBS operators and systems also within NTS itself.

8.1 Standardized MIDs for Traffic

Recent trends in the works of some PBBS authors indicate an awareness that message identification in general, not just the special case of Bulletin IDentification (BID), is the proper course to follow in our evolving network. STS supports this trend by having the ability to generate MIDs for traffic (T-MIDs) such that a message may freely transfer to and from the various components of NTS while maintaining the accountability MIDs provide. This MID is built from the preamble of the NTS message, not from external data, and thus, is reproducable at any stage. The format used by STS and proposed for general adoption is the letter T, the message number in the preamble, an underscore (_), and the callsign of the station of origin (without any extended originator data). Feedback from a bulletin posted in May proposing this plan indicated a potential problem if a message

must cross a previously traversed PBBS enroute its true destination. Analysis of this situation reveals that the message either has been re-addressed to a specific PBBS or it is starting a second tour of a routing loop that the sooner stopped the better. In consideration of this the author proposes the use of T-MIDs as outlined above for all NTS traffic on packet routed ZIP@NTS<state> and, if readdressed, any MID should be assigned by the station re-routing.

8.2 Extended Originator Field

Owing to the store and forward nature of packet message handling in conjunction with the growing popularity of personal mailboxes, increasingly, traffic originators may best be contacted via packet. In this light, provision for packet addressing data should aid especially the routing and distribution of service messages be they delivery advisories or failure reports. However, the address is useful only if available, and with the current state of directory services and percentage of the NTS operator population currently on packet, it is within reason to provide such data if available. This idea has potential drawbacks, and in the longer term, will require discussion and compromise. In the current release, STS supports the extended originator in the form call@PBBS.st.nation if the needed data is provided. As a point of departure for the evolution of this proposal stipulate that this format is transmitable via packet, ASCII, and voice. Further, that one or more of the characters used do not exist in the character sets used for CW, RTTY(U.S. Baudot or CCITT Number 2), and AMTOR and thus require a translation of some form. Let an additional point of agreement be to limit the highest domain level used to the national domain (e.g. usa, can, etc.). There are two characters that require translation or other handling @ and #, for @ the author proposes to substitute a pair of fraction (slant) bars (i.e. //). The # character used by some as a subdomain flag is unnecessary baggage and should be dropped as subdomains are easily located by position. There is also no reason that designations for current voice, CW, RTTY, and AMTOR nets could not be created to provide an equal degree of source indication for those operators and in any event the choice to include extended originator data is the option of any operator at the time of origination of each message. Continuing discussion is welcome and encouraged as the route to compromise.

9. DISTRIBUTION

The current primary methods of distribution are via MS-DOS magnetic media and downloading from various BBS's and PBBS's. The set of self-extracting LHarc archives used for PBBS/BBS distribution are the following: STSPC.COM the total package both source and MS-DOS run-time, STSPCX.COM is the MS-DOS run-time only distribution for those who do not need or want source code, STSSRC.COM is source code and support files only for compilation in other environments. If required the source and support files can be provided in cpio or tar formats with or without compression or packing by special arrangement with the author.

10. DISTRIBUTION PHILOSOPHY

The licensing and distribution terms for the STS package are the result of much thought. STS is distributed as a class of "freeware" with a measure of influence by the Free Software Foundation's (GNU) philosophy. This is NOT a work in the public domain and the license terms are such that STS is available to all who might need it for but the cost of media and shipping or a phone call. These terms boil down to - spread the word but you may not profit from it; you may only recover the out of pocket cost for media and postage and no more. Yes, you should make the program available to others who can make use of it, on a non-profit basis. Enjoy!

Comments on HF Digital Communications
Part 1 -- Link Level Issues

Tom Clark, W3IWI
6388 Guilford Road
Clarksville, MD 21029

1. Introduction: This paper and its companion Part 2. will discuss some of the issues involved in improving the throughput and reliability of amateur HF digital communications links. The intent is to present some strategies which should allow for a significant improvement with modest hardware and software investments.

As a member of the ARRL's "SKIPNET" HF "Special Temporary Authority" (STA), I feel very frustrated with the current situation. Even though the number of packet stations around the world have increased geometrically over the past 5 years, the ability of the long-haul HF networks to support the user-generated message traffic has been, at best, a "level" resource.

It is my opinion that there are two fundamental problems:

1. At the link level all digital modes are using rudimentary technology which grew out of standards which were merely convenient and which are far from optimum.

2. At the protocol level, AX.25 (as it is currently used) is incredibly inefficient and needs to be replaced.

I believe that developments in these two areas should proceed in parallel and will treat the topics separately.

2. The HF Radio "WIRE": It has often been stated that radio links are the worst possible "wires" for the carrying of data. This statement is especially true at HF. Modem designers using real "wires" are able to make three simplifying assumptions:

- The signal-to-noise ratio (SNR) is high and signals are stable for long periods.

- Any additive noise present in the system has a Gaussian probability distribution.

- Even though non-linearities may be present in the "wires", their effects are constant for long periods of time and may be handled by simple adaptive equalization.

Judged by these criteria, even our best VHF/UHF paths have inferior "wires", but at HF none of these criteria are applicable. And yet we attempt to use the wireline Bell 103 modem standards (200 Hz shift, 300 baud, one bit/baud) for HF packet and similar standards (170 Hz shift but with lower baud rates) for RTTY and AMTOR.

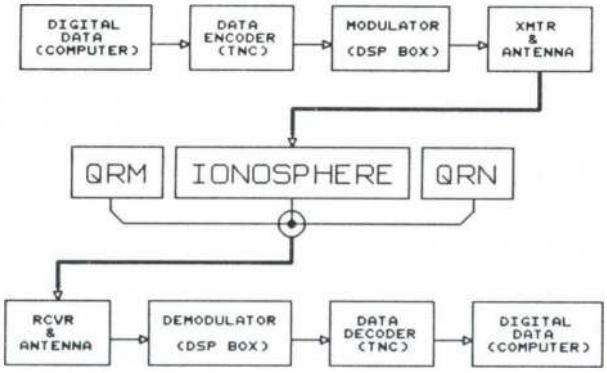
3. Modem Standards -- Old and New: The reason for using the 103-like standards is purely historical. In the early days 103 modems were available (and WB4APR picked up a large quantity on surplus). The 103 modem standards were easy to implement in the Exar XR2211 PLL demodulator in TAPR TNC-1 and TNC-2 and their clones. Manufacturers like AEA (with the PM-1 and later the PK-232) were able to use filter-based demodulators based on their RTTY "TU" products. The AMD AM7910 and TI TMS3105 single-chip "wireline" modems

provided inexpensive implementation paths for other manufacturers. But easy availability does not make the use of the simple modem technology optimum!

We will soon have Digital Signal Processing (DSP) hardware available. N4HY, W3IWI and others have been experimenting with TI TMS320-10 and -15 hardware for several years; this experimentation has led to several "products" which will be available in late 1990. The initial "offerings" will use the Motorola 56001 (AEA, DRSI and AMRAD) or TI 320-25 (TAPR/AMSAT) DSP chips. The AEA and DRSI hardware will be sold commercially, while AMRAD, TAPR and AMSAT are amateur development groups.

All these "products" are supposed to have well-defined "open architecture" so that many people can experiment with innovative modem ideas. When a new idea is available, the code will be distributed by telephone BBSes, Internet FTP file servers or over the air. I anticipate the early period of experimentation will involve membership in the "Modem of the Week Club"! The important thing to remember is that with either the 56001 or 320-25 we will own "engines" with computing power equivalent to a significant fraction of a Cray Y-MP (when doing well-defined specific tasks) and that our "Personal Cray" will have a cost less than the HF transceiver it is connected to.

4. The Data Path: In order to design an "optimum" modulation/demodulation system (if such even exists) we must consider the overall data transfer system, as outlined in this diagram:



This drawing is intentionally simplistic. The TNC-like data encoding/decoding functions may take place in the host computer or they may be in a separate box (for example, the TNC functions might be in the DSP box, or they may be in the host computer, or the functions may be split). Some data manipulation operations such as forward error correction (FEC) or convolutional encoding may best be handled by the DSP "engine". But for simplicity we shall consider the modulation/demodulation (MODEM) functions separate from coding functions.

In order to try to optimize the modem operations, it is necessary to consider the entire signal path. The cascaded effects of the radios, the antennas and the ionosphere can be considered to be a filter. An optimum demodulator needs to employ the conjugate filter to achieve optimum performance.

The radios and antennas are perhaps the simplest to deal with since their performance is nearly constant with time. James Miller, G3RUH has demonstrated that the combined inadequacies of FM transmitters and receivers can be improved by pre-distorting the transmitted waveform of 9600 BPS FSK signals. In principle, pre-distortion could be used at HF also.

Of more concern are the effects of ionosphere on the signals. Unfortunately radio signals propagated through the ionosphere exhibit considerable time variability and variability with

signal frequency. We shall return to this topic in subsequent sections.

The final area where the HF modem designer's life is made much more difficult arises because of the nature of the additive noise environment. While the wireline modems can assume a nearly Gaussian noise background, HF is plagued with impulsive noise (QRN) from thunderstorms, automobile ignitions, motor starting relays, and even the XYL's blender and garbage disposal. RF interference, both inadvertent and intentional (QRM) is a normal occurrence. The major defense against QRM and QRN is to insure that data protocols are tolerant of dropouts. It is also necessary to have receivers and demodulators which have good dynamic range and which recover rapidly from "glitches". Barry, VE3JF has also argued for the use of frequency diversity to help mitigate against such problems.

5. Bits vs. Bauds: In wireline applications we often see modulation schemes which jam upwards of 20,000 bits/sec through a 2 kHz wide telephone line. Many people *incorrectly* refer to V.32 modems as "9600 baud full-duplex" or TeleBit Trailblazers as handling up to 18 kbaud. It is often said "If such signals can go through a phone line, why don't we use them on the radio?"

The V.32 modems use a complex trellis coding scheme where an in-phase and quadrature phase channel are each amplitude modulated after the bits are convolutional encoded. The resulting complex waveform must handled very carefully if information is not to be lost. Typical $600 commercial V.32 modems use custom DSP chips with 18-bit A/D and D/A converters. The phone lines are equalized by a "training sequence" at the start of each transmission and assumed to be stable thereafter. Such are not the characteristics of our radio links!

The confusion between bits and bauds must be clearly understood. A bit is a piece of data. A baud represents an interval in time to accomplish the signaling.

As a simple example, consider the touch-tone telephone (DTMF) system. Each signaling element consists of one tone chosen from four low tones, and one of four high tones. The two one-of-four signals encode 16 possible states, equivalent to 4 bits. The same scheme could be expanded to having any number (1-8) of tones present, in which case there would be 255 possible states -- almost 8 bits worth of data (in this example, the zero state with no tones present is undefined).

For those who have studied Fourier transforms, a fundamental theorem of spectral analysis is that a frequency can be measured to an accuracy Δf in a time interval Δt related by the uncertainty principle

$$\Delta f \, \Delta t \approx \frac{1}{2\pi}$$

This is the *same* uncertainty principle encountered in quantum mechanics: the particle's position and momentum are uncertain at the level of Planck's constant.

In information theory the uncertainty principle is related to the Shannon limit which states that it takes at least one Hz of bandwidth to send one bit of data in one second. But you can "beat" the uncertainty principle if signals are strong, and a more correct statement is

$$\Delta f \, \Delta t \approx \frac{1}{2\pi * \mathrm{SNR}}$$

All of the modems which seem to violate the uncertainty principle and Shannon's limit do so by requiring stable signal and high SNR. One of the goals of HF modem development should be to figure out just which schemes can force bits through poor channels. Unfortunately, the FCC's amateur rules are very specific on what technology can be used. Much of the R&D work will require STA's.

6. The Ionosphere as a filter: Normally amateurs like to think of the ionosphere as a mirror and that their signals "bounce" off the mirror. However this is an overly simplistic view. In reality the ionosphere is a plasma permeated by a magnetic field and it is the electrons in this plasma which are responsible for radio propagation. The electron density below $\approx$100 km altitude is insignificant, and it rises to a maximum at $\approx$350-400 km. As a signal travels through this region it is continually bent. At any point along this path the index of refraction n (ignoring the effects of the magnetic field) is given by

$$n^2 = 1 - \frac{f_n^2}{f^2}$$

there f is the signal frequency and f_n is called the plasma frequency. The square of plasma frequency f_n^2 is in turn proportional to the *in situ* electron density N_e.

If a signal is sent vertically into the ionosphere and returns to the sender, then the signal must have penetrated the ionosphere to the level where the signal frequency f equals the plasma frequency f_n, at which point the index of refraction decreased to zero. The highest frequency at which this can occur represents the peak plasma frequency in the ionosphere. This occurs in the F_2 region and is often called f_oF_2, the critical frequency at vertical incidence. Vertical incidence radars called ionosondes are use to measure the profiles of electron density up to f_oF_2. Typically f_oF_2 will range from 1-2 MHz during solar minumum at nighttime to 12-14 MHz at solar maximum during the daytime. Occasionally (usually during the summer) patchy clouds with much higher electron densities (with f_n reaching as high as $\approx$ 50 MHz) will form in the E-region (100-120 km altitude) and give rise to sporadic-E propagation.

If the ionosphere is probed with a signal with $f > f_oF_2$, the vertical incidence signal will escape the ionosphere and travel off into space. However more oblique ray-paths will be gradually bent and may return to the earth. It is these signals which concern us here.

The gradual bending of signals on encountering an index which varies with height is easily seen visually on the desert or on a long stretch of paved road. The surface heating of the atmosphere in the first meter above the earth causes variations in the air's index of refraction which "reflect" (the proper word is *refract*) the sky as a visible mirage. The shimmering seen in the mirage will have its analog in multipath distortion of radio waves.

If one looks at the equation for the index of refraction carefully, it will be noticed that the index of refraction $n < 1$. In Physics were taught that the index of refraction n is the ratio

$$n = c/v$$

where c is the speed of light and v is the speed of the signal inside the medium. If $n < 1$ then it would seem that $v > c$ in contradiction to special relativity. To understand this paradox, we must understand the definition of the velocity of propagation in the medium. In the case of a plasma, the medium is dispersive, i.e. it's index of refraction changes with frequency. What happens is that the wavelength -- the distance between two points with the same phase --

shrinks in a plasma and the index of refraction could be expressed as:

$$n = \frac{\lambda_{plasma}}{\lambda_{vacuum}}$$

The velocity "measured" by the normal index of refraction is v_p, the *phase* velocity which is given by

$$v_p = \frac{1}{2\pi}\frac{\Phi}{f}$$

The rate at which information is transmitted is called the *group* velocity v_g and it is given by

$$v_g = \frac{1}{2\pi}\frac{d\Phi}{df}$$

which is the slope of the variation of phase vs. frequency. The group velocity $v_g < c$ and it is this velocity that is governed by special relativity. *Einstein is happy!*

7. On the air with the ionosphere -- how long is my baud? When we operate short paths near f_oF_2 (like on 80 meters) our signal undergoes many turns of phase shift as it is slowed down and turned around. All the little density variations in the ionosphere perturb the phase markedly and lead to the severe multipath distortion seen on 80M. Conversely when we operate long paths at oblique incidence angles (like on 10M or 15M long-haul links) the signal spends little time in the ionosphere so it has little distortion.

Back in 1980-81, this contrast was made very apparent in some BPSK tests we performed. Following the loss of the AMSAT Phase-3A spacecraft (when the Arianne launcher blew up), the Phase-3 telecommand team decided to try our 400 bits/sec Manchester-encoded PSK hardware on HF. The U.S. members of the team applied for and received an STA to use voice-bandwidth BPSK in the HF voice sub-bands (our foreign colleagues in Germany, Canada and New Zealand had much more liberal rules than we did). The first tests were between Ian, ZL1AOX and W3IWI on 10M; everything worked great. We exchanged digital data for hours on end even when signals were weak. Later tests between DJ4ZC, VE6SAT, WØPN and W3IWI on 20M worked pretty well also.

Then John, W1HDX near Boston and I tried 75M -- it was a disaster! Virtually no data could be exchanged despite good strong signals. We decided to try some simple bi-static radar tests to see why. With bi-static radar the transmitting and receiving stations are separated. Both John and I took our turn transmitting with the other listening. [*Lest the reader worry about the legality of radar transmissions on HF, what we actually sent was high speed CW with CW Identification. A MorseMatic keyer is a great pulse generator sending E E E E E at 99 WPM!*]

The receiving station generated a clock running at the same as the sender, and the received signals from an ordinary SSB receiver were observed on a scope synchronized to the independent receive clock. Lo and behold, the effects of multipath were discovered once more. The received signal would show an initial spike as the signals from the first path arrived. Then signals from other paths would arrive with random phases and the signal was all chopped up for a few msec. Then, after all the various wavefronts arrived the signal would stabilize. On the ≈600 km Massachusetts-to-Maryland path, it took 5-10 msec for the signal to stabilize. But our

400 BPS Manchester-encoded data had individual signaling elements only 1.25 msec long. Small wonder that our 75M PSK tests were unsuccessful.
RTTY and AMTOR operators on 80M and 40M know that 45 or 50 baud signals (about 20 msec per signaling element) usually works. But 110 baud ASCII (9.1 msec bits) is marginal. Very few have been successful with 300 baud Bell 103 packet transmission on 80M. Many military and commercial HF tests have similarly indicated that 50-75 baud is about the limit for HF paths shorter than $\approx$ 1000 km and frequencies below about twice the f_oF_2.

In the $\approx$ 1000-2000 km range covered by the 40M, 30M and 20M bands, clearly 300 baud does work -- witness the success of SKIPNET at moving large volumes of packet mail (albeit with low efficiency). But the AMTOR and RTTY stations running lower baud rates are able to move data when packet stations have difficulty. The strong evidence is that baud rates in the 100-200 range would be much more suitable when conditions are poor.

Since there are days when the ionosphere is good and days when it is horrid, an optimized strategy would be to have adaptive rates which change with conditions.

8. Coding and Data Rates: In the preceding discussion we talked in terms of BAUD rates -- the number of signaling elements that occur in one second. We now briefly mention some schemes which can (and will) be easily implemented in DSP hardware for testing. Which proves the most successful remains to be seen. In all of these schemes, each signaling element conveys more than one bit.

The first simple example is Quadrature Phase Shift Keying (QPSK). A carrier signal is phase modulated to one of four possible phases. Since there are four discrete states of the signal, then 2 bits of data can be encoded in each time slice. If the propagation medium is stable enough to permit 8 or 16 phase states (8PSK and 16PSK), then 3 or 4 bits could be conveyed. In addition to modulating the phase of the signals, QAM coding will be tested.

Commercial and military systems using multi-tone on/off keying (OOK) have been successful. We might design a scheme where 32 tones spaced 25 Hz apart are used, with one tone representing a bit. The 25 Hz channel spacing and the uncertainty principle discussed earlier mean that we can key the tones at 25 baud (40 msec signaling elements) and occupy about 850 Hz of spectrum. Although this is a 25 baud system, it conveys 800 bits/sec of data. Instead of using all 32 bits for data, we might choose to encode the data with an error correcting polynomial; depending on the degree of protection desired this might reduce the delivered data rate to $\approx$ 600-700 bits/sec. This multi-tone OOK approach, if carried to its limit, places severe limits on the dynamic range in transmitters; the peak-to-average power requirements are quite large. Since amateurs are notorious for "cranking up the wick" and reducing the peak-to-average ratio, on-the-air tests will be needed to see how gracefully multi-tone OOK looses performance and how "unfriendly" it is to users on nearby frequencies.

In the early days of packet radio, error correction was rejected because at the time the CPU chips in the VADG (8085), TNC-1 (6809) and TNC-2 (Z-80) lacked the "horsepower". The AMTOR protocols placed less stringent requirements on their embedded processors so FEC was included. Now the price of smart silicon has come down so the early decision should be revisited, especially for HF applications.

Comments on HF Digital Communications
Part 2 -- Data Protocol Issues

Tom Clark, W3IWI
6388 Guilford Road
Clarksville, MD 21029

1. Introduction: In part 1 we discussed some of the link-level issues in terms of the effects imposed by the ionosphere on HF signals and possible Digital Signal Processing (DSP) approaches to those issues.

In part 2 we assume that bits can be reliably delivered at reasonable speed and turn to questions like

- How can more information be crammed into each bit sent?
- How can the overhead associated with each transmission be minimized?
- How can the number of transmissions (and hence the number of times the overhead must be paid) be minimized?

2. Connected mode AX.25 -- the *WRONG* solution to the HF problem: Having watched literally thousands of attempts to move packet mail, I have become convinced that connected-mode AX.25 is a bad choice for use on HF. The U.S. stations on the HF nets operate under the ARRL "SKIPNET" STA (wherein the FCC authorizes unattended operation); most nets are "closed" with each station allowing connections only from other net members.

Despite the "closed" nets and well-equipped stations about half the attempts to forward mail result in timeouts due poor propagation and QRM. One of the most serious sources of QRM is from other net members. It appears to me that nets like 14109 kHz have sufficient activity that its channel capacity is reduced to the "ALOHA limit" of $\approx$ 18% which is shared among all the net members.

Messages longer than about 2 kbytes in size carried on the busier HF nets have a significant probability that they will result in a time-out and hence must be re-sent. The number of time they will need to be re-tried is proportional to the message size, and the time required for each attempt is also proportional to the message size. Therefore when a message exceeds a critical size, the channel time required to send that message will increase as the SQUARE of the message size. As a result many HF SYSOPs have adopted message size limits of 2-3 kbytes for traffic that will be handled.

Because of the combined difficulties associated with the ionosphere, plus QRM and QRN, plus the current modem technology used on HF (see Part 1.), a typical HF link has a bit error rate (BER) in the range of $1{:}10^2$ to $1{:}10^3$. Thus any packet frame longer than $\approx$ 500-1000 bits will probably not work; this has led to stations using PACLEN parameters in the range 40 to 80. Thus AX.25's per-frame overhead is about one-third of all bits sent.

In the present AX.25 protocol, if 4 frames are transmitted and the receiving station gets good copy on frames number 1, 3 and 4, then the inability of the protocol to reassemble frames requires frames 2, 3 and for to be resent. Because of this deficiency in the protocol, the typical HF SKIPNET station operates with MAXframe set to 1 or 2. The present protocol lacks any frame reassembly capability for historical reasons; the original TNCs circa 1983-84 didn't have enough computing capability or RAM to support the function.

Eric Gustafson, N7CL has developed an improved "PRIACK" modification to AX.25 which is now available for TNC-2 (and clones) and AEA TNCs. PRIACK gives channel priority to <ack> frames and uses p-persistent CSMA algorithms for sending I-frames. Despite PRIACK being available for nearly 2 years, it has not found wide acceptance. Many HF operators say "it slows down *MY* transmissions too much". Even if it were accepted, PRIACK is only a band-aid applied to an inappropriate protocol.

Another inefficiency (not intrinsic to AX.25) comes from the fact that all messages sent on HF are plain ASCII text and yet a full 8-bit byte is used to send the data. Only about 6.5 bits are needed for each character, corresponding to ≈ 20% loss of channel utilization. Even better would be to use data compression techniques (like .ZIP or .ARC) with full binary data transmission, which would make ≈ 50% improvement.

Add to all these factors the wasted keyup time for amplifiers (if used) on each end of the path required for each frame sent, and the time for the other station to send an <ack> and it becomes apparent why the real data throughput on HF channels is only a few tens of bits/second.

These factors may be summarized:

1. There is a need for new improved modem technology outlined in Part 1.

2. Radical protocol surgery is needed to solve the problems of timeouts, multiple re-transmission of messages, channel sharing, data compression, etc.

3. HEHFPRO -- Another Connectionless Protocol: HEHFPRO means "High Efficiency HF PROtocol. With HEHFPRO it is proposed to make use of unconnected AX.25 <UI> datagrams as an alternative to the present connected mode protocols.

Suppose that the W3IWI BBS has 23 messages to be sent to the European mail gateway at 4X1RU. W3IWI would collect all the messages into a single export file which might be 9132 bytes long. Although not required, for efficiency W3IWI compresses the first file with PKZIP into a new file 3932 bytes long. W3IWI then transmits a <UI> frame addressed to 4X1RU that says (in appropriate computerese):

```
Hello 4X1RU. W3IWI Calling. I have a binary .ZIPed mail file for you which is my number RU11367.
It is 3932 bytes long will be sent in blocks of 64 bytes/block. Let me know when you are ready.
```

If 4X1RU doesn't acknowledge W3IWI, the same <UI> frame is re-sent a few minutes later. When 4X1RU finally hears W3IWI, he responds with a response <UI> frame acknowledging the request. 4X1RU knows that the data portion of the blocks it receives subsequently will be the 64 data bytes long, plus 7 bytes to flag this data as a part of message RU11367, plus a two byte frame number or a total of 73 bytes long.

Since (3932 = 61*64 + 28), 4X1RU knows to expect a total of 62 such blocks and that the last block will have only 28 data bytes, and allocates space to hold the message. He prepares an response message with enough bits to flag each of the incoming blocks (in this case 8 bytes = 64 bits is the appropriate size) looking like

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000011
```

with 62 zeroes. This 8-byte field plus a few bytes of housekeeping information is sent as a <UI> frame response.

W3IWI then sends a suitable number of the 64-byte data blocks in one transmission, with each block corresponding to a separate <UI> frame with length 73 (including the message ID and the two-byte frame sequence

number). The number of <UI> frames sent in one transmission can be tailored to suit conditions, but lets assume 24 is chosen.

Of the first 24 frames sent, suppose that 4X1RU copies 14 frames numbered 0, 3-10, 14-17, 19, and 22 and would send a response <UI> frame with the 8-byte field now reading (from left to right):

10011111 11100011 11010001 00000000 00000000 00000000 00000000 00000011

The next tranmission would re-send the missing frames and append some new ones. After a few such attempts the acknowledge frame might look like:

11011111 11111111 11111111 11111101 11111111 11101111 11111101 11111111

with only frames numbered 2, 30, 43 and 54 missing. W3IWI would concentrate all efforts at filling in the missing pieces until an "all ones" response was received from 4X1RU. 4X1RU would then use PKUNZIP to de-compress the file and re-post the messages to their respective destinations on his "conventional" BBS.

At this point 4X1RU has informed W3IWI that he has the message complete. On receipt of this W3IWI would mark the outgoing file as delivered and initiate the <UI> datagram equivalent of a disconnect.

4. A few other design concepts: Even if no data transmissions were heard for a while from W3IWI, 4X1RU would continue to send <UI> frames indicating the current status of message RU11367 every few minutes so that when conditions permit data transfer can resume. If W3IWI encountered other activity on frequency, his software would enforce an automatic backoff. This process might take a few minutes or it might take hours but the messages would get through and there would be no such thing as a timeout.

Although a file containing mail was used for this example, the procedure is independent of data type, so it could also be used for file transfers. If the file is .ZIPped or .ARCed, then the file name is recovered on de-compression.

In this example we showed only one message file being transmitted. There is no reason that a given station could not have several sessions in progress at a given time in either direction. All "hooks" to implement these concepts (but not the HEHFPRO protocol) already exist in KA9Q's NOS software "engine".

Another potential application for an HEHFPRO-like protocol would be the amateur development of point-to-point meteor scatter links. The meteors in this case are not the "pings" associated with VHF DX operation which occur during meteor showers; rather they are the steady background of meteors which occur all the time. These meteors are most useful on forward scatter paths $\approx$ 800-1200 km long at frequencies like 6M and 10M. Under such conditions the meteor signals will rise from the noise for 1-2 seconds with little Doppler shift; such bursts will occur at a rate of about once per minute. To capitalize on such bursts, individual packet frames (blocks) must be shorter than about 0.5 seconds and it is desirable to be transmitting as much as possible to increase the probability of catching a meteor.

5. Acknowledgements: HEHFPRO has a lot of similarities to other protocols in use. In particular it draws heavily on TCP/IP's use of datagrams and I acknowledge KA9Q's patient tutorials on how things *should* be done. The main difference is that HEHFPRO has, in essence, a much longer sliding window and more extensive use of frame reassembly.

The use of a series of response bytes, with each bit corresponding to a fixed portion of the associated data/message, has been extensively used to load data and DCE messages into the UoSAT spacecraft for a number of years. Packet mail is handled by the DCE gateways by the same file export/import route described here.
GØ/K8KA and NK6K have prepared detailed specifications for a similar datagram protocol to be used with the PACSAT experiments on UoSAT and MICROSAT satellites. The main difference between HEHFPRO and the satellite protocols is that all terrestrial users in the satellite's footprint can hear the satellite and the satellite can pace data transfers over full-duplex links. On HF networks, all stations are presumed to be peers and the half-duplex links are plagued with "hidden terminal" problems.

I'd also like to offer special thanks to N4HY for serving as a sounding board for many if the ideas in both parts one and two of this tome.

BULLPRO

A Simple Bulletin Distribution Protocol

Tom Clark, W3IWI
6388 Guilford Road
Clarksville, MD 21029

1. INTRODUCTION: This paper proposes a simple protocol for the efficient local distribution of bulletins by packet radio.

One of the interesting changes that packet radio has brought to the amateur community is the rapid national and international distribution of information for the entire amateur radio community. The introduction of the BID (Bulletin Identification Designator) by WA7MBL and its subsequent implementation by all the major BBS software writers has made it possible for a bulletin posted on a PBBS in one area to wend its way to hundreds of other PBBSes in a day or two. This capability has been used to provide wide and rapid circulation of AMSAT and ARRL bulletins, DX news, hamfest announcements as well as providing for a national "soapbox" for individuals. This paper will address neither the sociological implications of this capability nor problems associated misaddressed bulletins, duplicates and corrupted BIDs.

The intent of this contribution is to propose a more efficient distribution scheme for the local user in his local area network (LAN). Presently the local user gets his personal copy of each bulletin by logging onto his local PBBS, scanning the topics of interest and then reading particular bulletins. Other users in the area repeat the process, and a particular bulletin may be read many times.

The act of fetching bulletins is left to the end user who must log into his local BBS and manually initiate the read request for each bulletin he wants to read. Joe Kasser's (G3ZCZ/W3) *LAN-LINK* software frees the user from this manual operation by logging onto area PBBSes automatically to fetch personal mail and bulletins. Other individuals set up personal PBBSes (sometimes called Personal Mailboxes) so that the material is forwarded automatically. Whether the end user read the bulletins manually or lets his computer do the work, bulletins are transmitted repeatedly for each user; such activity consumes a large fraction of the available Local Area Network (LAN) channel time.

2. BROADCAST DATAGRAM PROTOCOLS (BDP): The obvious solution to this wasteful use of LAN resources is for the bulletin to be broadcast to many users at the same time. Conceptually, one LAN "superstation" sends the bulletins as unconnected packet frames (called <UI> frames or datagrams) and other stations in the LAN monitor the <UI> frames. In the AX.25 protocol <UI> frames are addressed from the originating station to a specific address (the UNPROTO address for most TNCs) and they are CRC checked for validity. Since they are broadcast to many users, individual recipients do not <ack> the individual datagrams.

By definition, <UI> datagrams are un-numbered and have no implicit sequencing. Any BDP must have added sequencing information for proper reassembly of the bulletin. In addition it is necessary to transmit some additional information so that the receiving station knows when the bulletin has been received in entirety.

3. BULLPRO -- A SPECIFIC BDP: In designing BULLPRO I had several objectives:

- The protocol should be simple enough so that a minimal user with only a TNC and line printer could make use of it.

- If an end-user does have a computer, then he should be able to use simple utilities he already has (for simplicity this document assumes a PC-clone running MS-DOS and some capture-to-disk terminal program).

- The BULLPRO protocol could easily be expanded so that "smart" software could handle tasks like eliminating duplicates, automatically filing the bulletins and even requesting "fills" of missing information.

To make BULLPRO work, the transmitted bulletins must meet the following criteria:

A. Bulletins are line-oriented with each line terminated by a <cr> . The TNC used to send the bulletins is set up with PACLEN at least 20 greater than the longest line. The TNC operates in CONVerse mode with <cr> used as the SENDPAC trigger to send a frame. Thus each line of text corresponds to a separate <UI> frame.

B. Bulletins are N lines long and N < 1000.

C. Bulletins are identified by a unique character string, assumed to be a standard PBBS BID (the BID is 1-12 alphanumeric characters, upper case only. Blanks, the punctuation characters @ < $ > and control characters are reserved and should not be used. The BID string is then prepended with a $ identifier.).

Consider the following N=13 line long ARRL Bulletin which was distributed with the packet BID $ARLBØ26:

```
QST DE W1AW
ARRL BULLETIN 26  ARLBØ26
FROM ARRL HEADQUARTERS
NEWINGTON CT  JULY 3Ø, 199Ø
TO ALL RADIO AMATEURS

ON JULY 27 THE ARRL FILED ITS LEGAL BRIEF IN THE MATTER OF THE
PROPOSED REALLOCATION OF 22Ø TO 222 MHZ TO PRIVATE LAND MOBILE
SERVICES, FCC PR DOCKET 89-552, WITH THE UNITED STATES COURT OF
APPEALS FOR THE DISTRICT OF COLUMBIA CIRCUIT.  ORAL ARGUMENTS BEFORE
THE COURT ARE SCHEDULED FOR NOVEMBER 16, 199Ø.  THE CASE IS KNOWN AS
AMERICAN RADIO RELAY LEAGUE VERSUS FEDERAL COMMUNICATIONS COMMISSION
AND THE UNITED STATES OF AMERICA.
```

The bulletin distribution station using BULLPRO would send the bulletin by sending a beacon header with the following TNC commands:

```
MYCALL W3OBS-7                (as appropriate)
UNPROTO BULLTN VIA K9DOG
BEACON EVERY 6Ø
```

and then identifying the bulletin currently being transmitted by sending an additional line of text (Line#ØØ) which conveys the necessary descriptive material. This is done by setting setting the broadcast bulletin BTEXT to:

```
$ARLBØ26 L:ØØ/13  SB ALL @ ARRL  < W3OBS  9ØØ8Ø3  22Ø MHZ BRIEF FILED
```

where at least one blank separates the fields. The L:ØØ/13 identifies this as line zero with 13 more lines to follow (a total of 14 lines). If a bulletin is more than 100 lines long, the L: length field would be like L:ØØØ/234. The additional information in Line #ØØ gives the user the same information he would have copied from his local PBBS and is adequate to re-introduce the bulletin into the packet system. The BEACON EVERY xx time interval should be chosen so that the Line #ØØ identification beacon is sent several times while the bulletin is being transmitted by BULLPRO.

The W3OBS-7 bulletin server's software then meters out bulletin text at a rate of one line (one frame) every 1Ø-2Ø seconds (as appropriate to local conditions) and prepends sequencing information at the start of each line. The pattern of blanks in the prepended information should match that in Line #ØØ so that the most elementary character-oriented sort utilities (like MS-DOS SORT) can reassemble the bulletin.

```
$ARLBØ26 L:Ø1 QST DE W1AW
$ARLBØ26 L:Ø2 ARRL BULLETIN 26  ARLBØ26
$ARLBØ26 L:Ø3 FROM ARRL HEADQUARTERS
$ARLBØ26 L:Ø4 NEWINGTON CT  JULY 3Ø, 199Ø
$ARLBØ26 L:Ø5 TO ALL RADIO AMATEURS
$ARLBØ26 L:Ø6
$ARLBØ26 L:Ø7 ON JULY 27 THE ARRL FILED ITS LEGAL BRIEF IN THE MATTER OF THE
$ARLBØ26 L:Ø8 PROPOSED REALLOCATION (etcetera)
```

4. USER IMPLEMENTATION: At the user end, the minimal user can at least read the text and manually re-sequence it without ever logging onto the local PBBS.

A more sophisticated user could leave disk capture on overnight and save everything sent by W3OBS-7. An off-the-shelf utility like MSDOS's SORT could then be used to collect all lines with the BID $RLBØ26 together in order, and the FIND utility could be used to extract each bulletin into a separate file.

Assuming that bulletins are retransmitted several times, the user would be responsible for handling duplicated lines. The next step in sophistication would be to develop software to do this, to strip off the prepended sequence information, to maintain a list with the status of receipt of different BIDs, and to file the bulletins based on their BID or other Line#ØØ criteria.

If copying the BULLPRO broadcast bulletins proves unreliable, then an additional feature could be added. The bulletin servers could listen for <UI> datagrams which request a specific line to be re-sent, something like:

```
K9DOG>REQBUL:??? $ARLBØ26 L:Ø5
```

If a LAN has multiple BULLPRO servers, the user has the option of either selecting one with the MTO/MFROM or BUDLIST/LCALLS options in his TNC, or of accepting the multiple inputs and sorting out the duplicates in user's software. The latter option requires that all the servers adhere to a common BID standard. The uniform BID requirement is no more stringent than that imposed by the PBBSes to eliminate duplicates now.

Some comments on the "H"ierarchical Continent Address Designator

Tom Clark, W3IWI
6388 Guilford Road
Clarksville, MD 21029

At the risk of opening Pandorra's Box, this note suggests a change in the 2-character continent designator portion of the PBBS "H" hierarchical address field. An example would be the North America .NA portion of the packet address,

K9DOG @ W6SIX.DE.USA.NA

Let me state at the outset that I'm not convinced that we need to use the continent field. It seems to me that the country field by itself is adequate and that the packet BBSes can easily keep track of all the countries in the world. Let me also state that many of the issues addressed here arise because of constant confusion between the functions of addressing and routing.

The correct continent to assign to many countries of the world is confusing and/or ambiguous. To cite some examples of problems which have already been identified:

- 5B4=Cyprus in the Mediterranian is listed by the IARU (for WAC) and the ITU as Asia. Ditto 4X=Israel and JY=Jordan in the middle East.

- Both TA=Turkey and and the Soviet Union have part of their counties in Europe and part in Asia. 8Q6=Maldives is listed as both Africa and Asia. Several other countries are also "split".

- Although DU=Philipines and YB=Indonesia are regarded as Asiatic countries, the are listed as Oceania, along with Australia and Hawaii. If you venture to 9M=Malasia, you may be in either Oceania or Asia.

- The Central American and Carribean countries are nearly all part of North America, including YVØ, but not 9Y. Anyone care to guess what continent the southern part of HP=Panama is in?

If the purpose of the continent portion of the "H"ierarchical address is to facilitate delivery of messages, then it is illogical to route Israeli and Jordanian traffic through the Orient. It is illogical to make automated routing decisions for messages to Turkish amateurs based on whether the addressee is on the east or west side of the Straits of Bosporus.

Stations in Israel and Cyprus already face this dilemma. Rather than using a .AS Asian address, they choose to use .EU European designator to avoid having their packet mail routed via the Orient. Some have suggested the use of a new continent designator other than .AS; One suggestion has been .ME (Middle East) but this has a serious conflict with the state of Maine.

The Central Americans and the Carribean are both "legally" .NA but feel they need a separate geographic identity and both areas have independently suggested the use of .CA but this would conflict with .CA state.

All this leads to my suggestion that the present 2-letter continent designator must be changed. Either:

(1). The present 2-character designator should be dropped because it it not really needed and there are too many ambiguities. Users will always try to use address quirks to force routing, so don't give them the chance to fould things up. The computers at the international mail gateways can easily handle the entire DXCC list.

- or -

(2). A new logical regional designator which allows sub-continent sized regions should be adopted.

If a new, more flexible scheme is to be adopted, I'd suggest that new 4-character designators be chosen:

- .NOAM, .SOAM, .CEAM, .CARB replacing the present .NA & .SA and solving the Central American and Carribean problem,
- .ASIA replacing .AS for the Orient,
- .MDLE for the middle-eastern countries like 4X, JY etc.,
- Oceania divided into smaller areas like .NPAC, .SPAC, .AUST,
- The Indian ocean (now partly in .OC and partly in .AF) be designated .INDI,
- .AFRI replacing .AF for Africa and .EURO replacing .EU for Europe,
- .ANTR added for Antarctica,
- Additional new designators added as needed for sub-continent sized logical areas.

This scheme affords the logic of a 2-character field (.MD) for the "state", 3 characters (.DEU) for the country, & 4 (.AFRI) for the continent/subcontinent and avoids conflicts between state-sized areas and continents. Who knows, a few decades hence a 5 character field (.EARTH, .VENUS) may be needed too!

[Note: Because of issues raised in this note, the W3IWI PBBS does not use a continent designator on any of its own transactions, although it transparently passes any originated elsewhere].

PUBLISHED BY THE AMERICAN RADIO RELAY LEAGUE

NEWINGTON, CONNECTICUT